



[12] 发明专利申请公开说明书

[21] 申请号 03152256.4

[43] 公开日 2004 年 3 月 17 日

[11] 公开号 CN 1482540A

[22] 申请日 2003.8.1 [21] 申请号 03152256.4

[30] 优先权

[32] 2002. 8. 2 [33] JP [31] 226682/2002

[71] 申请人 松下电器产业株式会社

地址 日本大阪府

[72] 发明人 瓶子岳人 坂田俊幸 小川一
宫地凉子 宫阪修二 石川智一

[74] 专利代理机构 永新专利商标代理有限公司

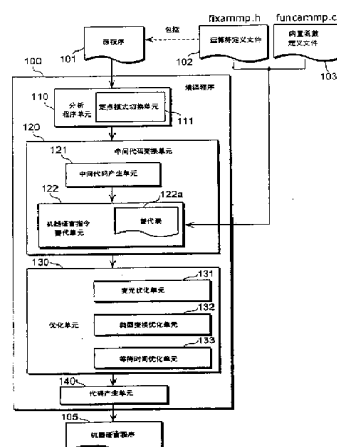
代理人 韩 宏

权利要求书 11 页 说明书 54 页 附图 71 页

[54] 发明名称 编译程序、编译程序装置和编译方法

[57] 摘要

提供了在源程序 101 中包括的运算符定义文件 102 等和将源程序 101 翻译成机器语言程序 105 的编译程序 100。运算符定义文件 102 包括由类定义对各种定点类型运算符的定义。编译程序 100 可以有效地产生处理器执行的高级和专用指令，并通过扩展函数等来作出改进，而不频繁地重复编译程序本身的版本的更新。编译程序 100 由产生中间编码的中间编码产生单元 121、用机器语言指令替代引用由运算符定义文件 102 定义的类的中间编码的机器语言指令替代单元 122 和执行以包括替代的机器语言指令的中间编码为目标的优化的优化单元 130 组成。



1.一种将一个源程序翻译成一个机器语言程序的编译程序，所述机器语言程序包括其中定义了对应于一个专用于一个目标处理器的机器语言指令的运算的运算定义信息，编译程序包括：

分析程序步骤，分析源程序；

中间编码变换步骤，将分析的源程序变换成中间编码；

优化步骤，优化变换的中间编码；以及，

代码产生步骤，将优化的中间编码变换成机器语言指令，

其中，中间编码变换步骤包括：

检测子步骤，检测任何中间编码是否引用在运算定义信息中定义的运算；以及，

替代子步骤，当检测到中间编码时，用一个对应的机器语言指令替代中间编码，以及，

在优化步骤中，优化中间编码，中间编码包括在替代子步骤中替代中间编码的机器语言指令。

2.如权利要求1所述的编译程序，

其中，运算定义信息是要在源程序中包括的主文件，

在主文件中，运算由一个由数据和方法组成的类定义，以及

在中间编码变换步骤中，通过检测任何中间编码是否引用由主文件定义的类来检测任何中间编码是否引用运算。

3.如权利要求2所述的编译程序，

其中，所述类定义一个定点类型，以及

在检测子步骤中，检测使用定点类型数据的中间编码。

4.如权利要求 3 所述的编译程序，

其中，所述类中的方法定义以定点类型数据为目标的运算符，

在检测子步骤中，检测是基于以一个运算为目标的运算符和数据类型的一个集合是否适合所述方法中的定义来执行的，以及

在替代步骤中，其运算符和数据类型的集合适合该定义的中间编码被用一个对应的机器语言指令来替代。

5.如权利要求 2 所述的编译程序，

其中，所述类定义一个 SIMD 类型，以及

在检测子步骤中，检测使用 SIMD 类型数据的中间编码。

6.如权利要求 5 所述的编译程序，

其中，所述类中的方法定义了以 SIMD 类型数据为目标的运算符，

在检测子步骤中，检测是基于以一个运算为目标的运算符和数据类型的一个集合是否适合所述方法中的定义来执行的，以及

在替代步骤中，其运算符和数据类型的集合适合该定义的中间编码被用一个对应的机器语言指令来替代。

7.如权利要求 2 所述的编译程序，

其中，所述类与实现对应的处理的一个机器语言指令相联系，
以及，

在替代子步骤中，中间编码被用与所述类相联系的一个机器语言指令替代。

8.如权利要求 2 所述的编译程序，

其中，所述类与实现对应的处理的两个或更多机器语言指令相联系，以及，

在替代子步骤中，中间编码被用与所述类相联系的两个或更多机器语言指令替代。

9.如权利要求 1 所述的编译程序，

其中，运算定义信息是在源程序中包括的主文件，

在主文件中，运算由所述函数定义，以及

在中间编码变换步骤，通过检测任何中间编码是否引用由主文件定义的函数来检测任何中间编码是否引用运算。

10.如权利要求 9 所述的编译程序，

其中，所述函数描述实现对应处理的一个机器语言指令，以及

在替代子步骤中，中间编码被用在函数中描述的一个机器语言指令替代。

11.如权利要求 10 所述的编译程序，

其中，所述函数包括一个返回被表示为从输入数据的最高有效位开始的一系列 0 的位的个数的函数，以及

在所述函数中描述的机器语言指令对被表示为从存储在第一寄存器中的一个值的最高有效位开始的一系列 0 的位的个数进行计数，并将结果存储在第二寄存器中。

12.如权利要求 10 所述的编译程序，

其中，所述函数包括一个返回被表示为从最高有效位开始的一

系列 1 的位的个数的函数，以及，

在所述函数中描述的机器语言指令对关于存储在第一寄存器中的值的被表示为从最高有效位开始的一系列 1 的位的个数进行计数，并将结果存储在第二寄存器中。

13.如权利要求 10 所述的编译程序，

其中，所述函数包括一个返回与输入数据的最高有效位相同的值连续的位的个数的函数，以及，

在所述函数中描述的机器语言指令对由一系列与存储在第一寄存器中的值的最高有效位相同的值表示的位的个数进行计数，并将结果存储在第二寄存器中。

14.如权利要求 13 所述的编译程序，

其中，所述函数返回被表示为从最高有效位的下一位开始的一系列与输入数据的最高有效值相同的值的位的个数，以及，

在所述函数中描述的机器语言指令对被表示为从存储在第一寄存器中的值的最高有效位的下一位开始的一系列与最高有效位相同的值的位的个数进行计数，并将结果存储在第二寄存器中。

15.如权利要求 10 所述的编译程序，

其中，所述函数包括一个返回在输入数据中包括的位 1 的个数的函数，以及，

在所述函数中描述的机器语言指令对存储在第一寄存器中的值的位 1 的个数进行计数，并将结果存储在第二寄存器中。

16.如权利要求 10 所述的编译程序，

其中，所述函数包括一个基于从输入数据在指定位位置提取的位来返回一个符号扩展值的函数，以及，

在所述函数中描述的机器语言指令从存储在第一寄存器中的值取出在由第二寄存器指定的位位置的位，对所述位进行符号扩展，并将符号扩展后的位存储在第三寄存器中。

17.如权利要求 10 所述的编译程序，

其中，所述函数包括一个基于从输入数据在指定位位置提取的位来返回一个零扩展值的函数，以及，

在所述函数中描述的机器语言指令从存储在第一寄存器中的值取出在由第二寄存器指定的位位置的位，对所述位进行零扩展，并将零扩展后的位存储在第三寄存器中。

18.如权利要求 9 所述的编译程序，

其中，所述函数描述一个包括实现对应的处理的两个或更多机器语言指令的机器语言指令序列，以及

在替代子步骤中，中间编码被用机器语言指令序列替代。

19.如权利要求 18 所述的编译程序，

其中，所述函数包括一个更新模数寻址的地址的函数。

20.如权利要求 19 所述的编译程序，

其中，在所述函数中描述的机器语言指令序列包括一个在第三寄存器中存储一个通过用一个存储在第二寄存器中的值替代存储在第一寄存器中的值的预定位字段而获得的值的机器语言指令。

21.如权利要求 18 所述的编译程序，

其中，所述函数包括一个更新位反转寻址的地址的函数。

22.如权利要求 21 所述的编译程序，

其中，在所述函数中描述的机器语言指令序列包括一个在第三寄存器中存储一个通过逐位反转存储在第一寄存器中的值的预定位字段的位置而获得的值的机器语言指令。

23.如权利要求 9 所述的编译程序，

其中，所述函数包括一个可以用累加器作为基准类型来指定一个临时变量的函数，所述函数是一个同时更新不以优化中的分配为目标的累加器和以优化中的分配为目标的通用寄存器的操作。

24.如权利要求 23 所述的编译程序，

其中，所述函数执行一个乘法并可以用累加器作为基准类型来指定一个临时变量，累加器存储乘法的结果。

25.如权利要求 23 所述的编译程序，

其中，所述函数执行一个乘积和并可以用累加器作为基准类型来指定一个临时变量，累加器存储乘积和的结果。

26.如权利要求 9 所述的编译程序，

其中，在替代子步骤，引用所述函数的中间编码被一个具有对应于所述函数的各种变元的各种运算数的机器语言指令替代。

27.如权利要求 26 所述的编译程序，

其中，在替代子步骤，一个引用所述函数的中间编码 (i) 在所有变元是常数时被一个其运算数是通过保持在常数而获得的常数值值的机器语言指令所替代；(ii) 在变元的一部分是常数时被一个具有

立即值运算数的机器语言指令所替代；以及（iii）在所有变元是变量时被一个具有寄存器运算数的机器语言指令所替代。

28.如权利要求 1 所述的编译程序，

其中，优化步骤包括类型变换子步骤，在不同类型之间执行一个操作的多个中间编码或机器语言指令被执行所述操作的一个机器语言指令所替代。

29.如权利要求 28 所述的编译程序，

其中，在类型变换子步骤中，执行一个乘以两个 n 位变量并将结果存储在一个 $2n$ 位变量中的操作的多个中间编码或机器语言指令被执行所述操作的一个机器语言指令所替代。

30.如权利要求 29 所述的编译程序，

其中，在类型变换子步骤中，当向两个变量作出实现从 n 位到 $2n$ 位的类型变换的显式说明时，运算被机器语言指令所替代。

31.如权利要求 1 所述的编译程序，

其中，编译程序以一个具有两个或更多个执行一个以两个或更多定点类型为目标的运算的定点模式的处理器为目标，

在分析程序步骤，在源程序中检测一个切换定点模式的描述，以及，

编译程序进一步包括定点模式切换步骤，当在分析程序步骤检测到所述描述时插入一个机器语言指令以遵循切换定点模式的描述来切换定点模式。

32.如权利要求 31 所述的编译程序，

其中，切换定点模式的描述与一个目标函数相联系，以及
在定点模式切换步骤，用于保存和返回定点模式的机器语言指令被分别插入到对应的函数的头部和尾部。

33.如权利要求 1 所述的编译程序，

其中，优化步骤包括检测源程序中的描述的等待时间优化子步骤，所述描述指定其中在特定位置的执行时间只被保证预定数目的周期的等待时间，并调度一个机器语言指令以使得等待时间是依据所检测的指定被保证的。

34.如权利要求 33 所述的编译程序，

其中，在等待时间优化子步骤，当检测到以带有第一标号的第一机器语言指令和带有第二标号的第二机器语言指令之间的间隔为目标的指定预定周期的等待时间的描述时，执行时间调度以使得从执行第一机器语言指令开始直到执行第二机器语言指令只花费所述数目的周期的执行时间。

35.如权利要求 33 所述的编译程序，

其中，在等待时间优化子步骤，当检测到以对一个规定寄存器的访问为目标的指定预定周期的等待时间的描述时，执行时间调度以使得从执行访问寄存器的机器语言指令开始直到下一次执行访问所述寄存器的机器语言指令只花费所述数目的周期的执行时间。

36.如权利要求 1 所述的编译程序，

其中，编译程序进一步包括一个类库，以便将用在运算定义信息中的机器语言指令以不同于所述编译程序作为目标的第一处理器

的第二处理器的机器语言指令替代。

37.一种计算机可读记录介质，在其上记录了在要编译的源程序中包括的主文件，

其中，其中定义了对应于专用于目标处理器的机器语言指令的运算的运算定义信息是在源程序中包括的主文件，

在主文件中，运算由一个由数据和方法组成的类来定义。

38.一种计算机可读记录介质，在其上记录了在要编译的源程序中包括的类库，

其中，编译程序进一步包括一个类库，以便将用在运算定义信息中的机器语言指令以不同于所述编译程序作为目标的第一处理器的第二处理器的机器语言指令替代。

39.一种计算机可读记录介质，在其上记录了要编译的包括主文件或类库中的至少一个的源程序，

其中，其中定义了对应于一个专用于目标处理器的机器语言指令的运算的运算定义信息是在源程序中包括的主文件，

在主文件中，运算由一个由数据和方法组成的类来定义，以及

编译程序进一步包括一个类库，以便将用在运算定义信息中的机器语言指令以不同于所述编译程序作为目标的第一处理器的第二处理器的机器语言指令替代。

40.一种将一个源程序翻译成一个机器语言程序的编译程序装置，编译程序装置包括：

可用于保存其中事先定义了对应于一个专用于一个目标处理器

的机器语言指令的运算的运算定义信息的单元，

分析程序单元，可用于分析源程序；

中间编码变换单元，可用于将分析的源程序变换成中间编码；

优化单元，可用于优化变换的中间编码；

代码产生单元，可用于将优化的中间编码变换成机器语言指令，

其中，中间编码变换单元包括：

检测单元，可用于检测任何中间编码是否引用在运算定义信息中定义的运算；以及，

替代单元，当检测到中间编码时，可用于用一个对应的机器语言指令替代中间编码，以及，

优化单元用包括在替代单元中替代的机器语言指令的中间编码执行优化。

41.一种将一个源程序翻译成一个机器语言程序的编译方法，包括：

分析程序步骤，分析源程序；

中间编码变换步骤，将分析的源程序变换成中间编码；

优化步骤，优化变换的中间编码；以及，

代码产生步骤，将优化的中间编码变换成机器语言指令，以及

其中，中间编码变换步骤包括：

检测子步骤，检测中间编码中的任一个是否引用在其中事先定义了对应于一个专用于一个目标处理器的机器语言指令的运算的运算定义信息中定义的运算；

替代子步骤，当检测到中间编码时，用一个对应的机器语言指令替代中间编码，以及，

在优化步骤中，优化中间编码，中间编码包括在替代子步骤中替代中间编码的机器语言指令。

编译程序、编译程序装置和编译方法

技术领域

本发明涉及一种将以诸如 C++ 语言的高级程序语言描述的源程序编译成机器语言的编译程序，尤其涉及编译程序的优化。

背景技术

随着处理器的功能在近些年来被极大地提高，强烈地寻求可以高效地开发处理器所具有的高度功能的高性能编译程序。换句话说，需要高效地产生由目标处理器执行的高级和专用指令的编译程序。

例如，开发了执行媒体处理、例如数字信号处理所需的各种定点格式的运算指令的处理器和执行 SIMD（单指令多数据）类型指令的高性能处理器。需要以这样一个处理器为目标的编译程序通过高效地产生各种定点格式的运算指令和 SIMD 类型指令来优化代码大小和执行速度。

然而，无须说明的是，一个常规编译程序高效地产生处理器执行的关于以诸如 C++ 语言的高级程序语言描述的源程序的高级和专用指令。因此，在用于媒体处理的应用和在代码大小和执行速度方面需要严格条件的其他应用的开发中，用户在当前条件下别无选择地只能描述汇编程序指令中的临界点。但存在一个问题，汇编程序指令的程序设计不仅需要更多的工时，在可维护性和可移植性上与

采用诸如 C++语言的高级程序语言的开发相比还非常低级。

另外，常规编译程序在其自身中具有用于产生处理器执行的高级和专用指令等的优化处理。换句话说，高效利用目标处理器的特征来用于优化的处理模块被包括在编译程序自身内并集成。因此，当编译程序的功能被扩展或目标编译程序的说明书改变时，需要重新配置整个编译程序。这存在一个问题，必须每次重复编译程序的版本的升级等。

发明内容

考虑到上述问题，本发明的第一个目的是提供一个可以高效地产生处理器执行的高级和专用指令的编译程序。

此外，本发明的第二个目的是提供一个可以在不频繁地重复编译程序自身的版本的升级的情况下通过扩充功能等作出改进的编译程序。

依据本发明的编译程序将一个源程序翻译成一个机器语言程序，该程序包括运算定义信息，其中定义了对应于一个专用于一个目标处理器的机器语言指令的运算，编译程序包括：分析程序步骤，分析源程序；中间编码变换步骤，将分析的源程序变换成中间编码；优化步骤，优化变换的中间编码；以及，代码产生步骤，将优化的中间编码变换成机器语言指令，其中，中间编码变换步骤包括：检测子步骤，检测中间编码中的一个是否引用（refer to）在运算定义信息中定义的运算；以及，替代子步骤，当检测到中间编码时，用

一个对应的机器语言指令替代中间编码，以及，在优化步骤中，优化中间编码，中间编码包括在替代子步骤中替代中间编码的机器语言指令。

例如，依据本发明的程序由源程序中包括的主文件（header file）和将源程序翻译成机器语言程序的编译程序组成；在主文件中，定义了一个由数据和方法组成的类；编译程序包括：分析程序步骤，分析源程序；中间编码变换步骤，将分析的源程序变换成中间编码；优化步骤，优化变换的中间编码；以及，代码产生步骤，将优化的中间编码变换成机器语言指令，其中，中间编码变换步骤包括：检测子步骤，检测中间编码中的一个是否引用在主文件中定义的类；以及，替代子步骤，当检测到中间编码时，用一个对应的机器语言指令替代中间编码，以及，在优化步骤中，优化中间编码，中间编码包括在替代子步骤中替代中间编码的机器语言指令。

从而，当有一个语句引用在源程序中的主文件中定义的类时，在中间编码被变换成机器语言指令之后对应于该语句的中间编码变成优化处理的目标，因此，中间编码可以与附近的机器语言指令一起被优化。另外，由于编译程序不仅由编译程序自身的功能性能执行优化（优化处理）而且与主文件中的定义相结合来执行优化，所以编译程序可以增加作为优化目标的语句并提高优化水平。

这里，可接受的是该类定义一个定点类型，并且，在检测子步骤中，检测使用定点类型数据的中间编码，并且，仍然可接受的是在类中的方法定义了以定点类型数据为目标的运算符，在检测子步

骤中，检测是基于以一个运算为目标的一个运算符和数据类型集合是否适合方法中的定义来执行的，以及，在替代步骤中，其运算符和数据类型的集合适合定义的中间编码被用一个对应的机器语言指令来替代。

从而，由于由主文件定义的定点类型和运算符通过编译程序与普通类型类似地通过简单地在源程序中包括主文件而与普通类型类似地被变换成对应的中间编码和机器语言指令，所以用户可以声明和使用对应于专用于目标处理器的定点模式的类型。

此外，依据本发明的编译程序包括在源程序中包括的主文件并将源程序翻译成机器语言程序的编译程序；在主文件中定义一个函数（function）；编译程序包括：分析程序步骤，分析源程序；中间编码变换步骤，将分析的源程序变换成中间编码；优化步骤，优化变换的中间编码；以及，代码产生步骤，将优化的中间编码变换成机器语言指令，其中，中间编码变换步骤包括：检测子步骤，检测中间编码中的一个是否引用在主文件中定义的函数；以及，替代子步骤，当检测到中间编码时，用一个对应的机器语言指令替代中间编码，以及，在优化步骤中，优化中间编码，中间编码包括在替代子步骤中替代中间编码的机器语言指令。

从而，当有一个语句引用在源程序中的主文件中定义的函数（内置函数）时，在中间编码被变换成由主文件定义的机器语言指令之后，对应于该语句的中间编码变成优化处理的目标，因此，中间编码可以与附近的机器语言指令一起被优化。另外，当用户想要使用

专用于处理器的高函数指令（high-functional）时，他仅仅需要描述出主文件被包括在源程序中并且在源程序中调用所需要的内置函数。换句话说，将他从利用汇编程序指令编码中释放出来。

如上所述，采用依据本发明的编译程序，高效地产生目标处理器所执行的高函数和专用指令；以高水平执行优化；以及，通过主文件的灵活响应、例如函数扩展变为可能，因此，编译程序的实用价值非常高，尤其是作为用于需要代码大小和执行速度上的严格说明的媒体处理应用的开发工具。

应该注意，本发明不仅可以实现为类似这个的编译程序，也可以实现为采用在用于编译程序的程序中包括的步骤作为步骤的编译程序装置和其中记录有特征化的编译程序或主文件的计算机可读记录介质。然后，无须说明的是，类似这些的程序和数据文件可以通过诸如 CD-ROM 的记录介质或诸如因特网的传输媒介被广泛地分布。

作为关于这个申请的技术背景的进一步的信息，在 2002 年 8 月 2 日申请的日本专利申请 No.2002-33668 在这里被包含来作为参考。

附图说明

从下面结合附图的描述中，本发明的这些和其他目的、优点和特征将变得明显，其中附图显示了本发明的一个具体实施例。在附图中：

图 1 是显示依据本发明的编译程序的目标处理器的示意方框图。

图 2 是显示处理器的算术和逻辑/比较运算单元的示意图。

图 3 是显示处理器的桶形移位器的配置的方框图。

图 4 是显示处理器的变换器的配置的方框图。

图 5 是显示处理器的除法器的配置的方框图。

图 6 是显示处理器的乘积运算单元的乘法/求和的配置的方框图。

图 7 是显示处理器的指令控制单元的配置的方框图。

图 8 是显示处理器的通用寄存器（R0～R31）的配置的方框图。

图 9 是显示处理器的连接寄存器（LR）的配置的示意图。

图 10 是显示处理器的分支寄存器（TAR）的配置的示意图。

图 11 是显示处理器的程序状态寄存器（PSR）的配置的示意图。

图 12 是显示处理器的条件标志寄存器（CFR）的配置的示意图。

图 13A 和 13B 是显示处理器的累加器（M0, M1）的配置的示意图。

图 14 是显示处理器的程序计数器（PC）的配置的示意图。

图 15 是显示处理器的 PC 保存寄存器（IPC）的配置的示意图。

图 16 是显示处理器的 PSR 保存寄存器（IPSR）的配置的示意图。

图 17 是显示处理器的流水线行为的时序图。

图 18 是显示在执行一个指令时的处理器的流水线行为的每个阶段的时序图。

图 19 是显示处理器的并行行为的示意图。

图 20 是显示由处理器执行的指令的格式的示意图。

图 21 是解释一个属于类别“ALUadd（加法）系统”的指令的示意图。

图 22 是解释一个属于类别“ALUsub（减法）系统”的指令的示意图。

图 23 是解释一个属于类别“ALUlogic（逻辑运算）系统等”的指令的示意图。

图 24 是解释一个属于类别“CMP（比较运算）系统”的指令的示意图。

图 25 是解释一个属于类别“mul（乘法）系统”的指令的示意图。

图 26 是解释一个属于类别“mac（乘积和运算）系统”的指令的示意图。

图 27 是解释一个属于类别“msu（乘积差）系统”的指令的示意图。

图 28 是解释一个属于类别“MEMld（从存储器装入）系统”的指令的示意图。

图 29 是解释一个属于类别“MEMstore（存储在存储器中）系统”的指令的示意图。

图 30 是解释一个属于类别“BRA（分支）系统”的指令的示意图。

图 31 是解释一个属于类别“BSasl（算术桶形移位）系统等”的指令的示意图。

图 32 是解释一个属于类别“BSIsr（逻辑桶形移位）系统等”的指令的示意图。

图 33 是解释一个属于类别“CNVvaln（算术变换）系统”的指令的示意图。

图 34 是解释一个属于类别“CNV（普通变换）系统”的指令的示意图。

图 35 是解释一个属于类别“SATvlpk（饱和处理）系统”的指令的示意图。

图 36 是解释一个属于类别“ETC（等等）系统”的指令的示意图。

图 37 是显示依据本发明的编译程序的配置的功能方框图。

图 38 是显示在运算符定义文件中的清单的一部分的示意图。

图 39 是显示在运算符定义文件中的清单的一部分的示意图。

图 40 是显示在运算符定义文件中的清单的一部分的示意图。

图 41 是显示在运算符定义文件中的清单的一部分的示意图。

图 42 是显示在运算符定义文件中的清单的一部分的示意图。

图 43 是显示在运算符定义文件中的清单的一部分的示意图。

图 44 是显示在运算符定义文件中的清单的一部分的示意图。

图 45 是显示在运算符定义文件中的清单的一部分的示意图。

图 46 是显示在运算符定义文件中的清单的一部分的示意图。

图 47 是显示在运算符定义文件中的清单的一部分的示意图。

图 48 是显示在运算符定义文件中的清单的一部分的示意图。

图 49 是显示在运算符定义文件中的清单的一部分的示意图。

图 50 是显示在运算符定义文件中的清单的一部分的示意图。

图 51 是显示在运算符定义文件中的清单的一部分的示意图。

图 52 是显示在运算符定义文件中的清单的一部分的示意图。

图 53 是显示在运算符定义文件中的清单的一部分的示意图。

图 54 是显示在运算符定义文件中的清单的一部分的示意图。

图 55 是显示在运算符定义文件中的清单的一部分的示意图。

图 56 是显示在运算符定义文件中的清单的一部分的示意图。

图 57 是显示在运算符定义文件中的清单的一部分的示意图。

图 58 是显示在运算符定义文件中的清单的一部分的示意图。

图 59 是显示在运算符定义文件中的清单的一部分的示意图。

图 60 是显示在运算符定义文件中的清单的一部分的示意图。

图 61 是显示在运算符定义文件中的清单的一部分的示意图。

图 62 是显示在运算符定义文件中的清单的一部分的示意图。

图 63 是显示在运算符定义文件中的清单的一部分的示意图。

图 64 是显示在运算符定义文件中的清单的一部分的示意图。

图 65 是显示在运算符定义文件中的清单的一部分的示意图。

图 66 是显示在运算符定义文件中的清单的一部分的示意图。

图 67 是显示在运算符定义文件中的清单的一部分的示意图。

图 68 是显示在运算符定义文件中的清单的一部分的示意图。

图 69 是显示在内置函数定义文件中的清单的一部分的示意图。

图 70 是显示在内置函数定义文件中的清单的一部分的示意图。

图 71 是显示在内置函数定义文件中的清单的一部分的示意图。

图 72 是显示在内置函数定义文件中的清单的一部分的示意图。

图 73 是显示机器语言指令替代单元的行为的流程图。

图 74 是显示优化单元的变元优化单元的行为的流程图。

图 75 是显示一个算术树以解释优化单元的类型变换优化单元的行为的示意图。

图 76 是显示一个样本程序的例子以解释等待时间（latency）优化单元的行为的示意图。

图 77 是解释分析程序单元的定点模式切换单元的行为的示意图。

图 78 是解释采用一个类库（class library）的行为验证技术的示意图。

具体实施方式

下面采用附图详细解释依据本发明的本实施例的编译程序。

依据本实施例的编译程序是一个将以诸如 C/C++语言的高级程序语言描述的源程序翻译成特定处理器（目标）可以执行的机器语言程序的交叉编译程序，其特征是它可以指定要产生的机器语言程序的代码大小和执行时间精密相关的优化的命令。

（处理器）

首先，采用图 1 到图 36 解释依据本实施例的编译程序的目标处

理器的一个例子。

作为依据本实施例的编译程序的目标的处理器例如是一个已经开发的用在 AV 媒体信号处理技术领域的通用处理器，并且可执行指令与普通微计算机相比具有更高的并行性。

图 1 是显示本处理器的示意方框图。处理器 1 是一个运算（opentional）装置，其字长度是 32 位（一个字），由指令控制单元 10、译码单元 20、寄存器文件 30、运算单元 40、I/F 单元 50、指令存储器单元 60、数据存储器单元 70、扩充寄存器单元 80 和 I/O 接口单元 90 组成。运算单元 40 包括算术和逻辑/比较运算单元 41~43、乘法/乘积和运算单元 44、桶形移位器 45、除法器 46 和变换器 47 来用于执行 SIMD 指令。乘法/乘积和运算单元 44 能够处理最大 65 位的累加以使得不降低位精度。乘法/乘积和运算单元 44 还能够与在算术和逻辑/比较运算单元 41~43 的情况中一样执行 SIMD 指令。此外，处理器 1 能够在最大三个数据单元上并行执行算术和逻辑/比较运算指令。

图 2 是显示算术和逻辑/比较运算单元 41~43 的示意图。算术和逻辑/比较运算单元 41~43 中的每一个由 ALU 单元 41a、饱和处理单元 41b 和标志单元 41c 组成。ALU 单元 41a 包括算术运算单元、逻辑运算单元、比较器和 TST 组成。要支持的运算数据的位宽度是 8 位（一个字节。此时，并行使用四个运算单元）、16 位（半个字。此时，并行使用两个运算单元）和 32 位（一个字。此时，使用所有运算单元处理 32 位数据）。对于算术运算的结果，标志单元 41c 等

检测一个溢出并产生一个条件标志。对于运算单元、比较器和 TST 中的每一个的结果，执行算术右移、由饱和处理单元 41b 进行的饱和、最大/最小值的检测、绝对值产生处理。

图 3 是显示桶形移位器 45 的配置的方框图。桶形移位器 45 由选择器 45a 和 45b、高位移位器 45c、低位移位器 45d 和饱和处理单元 45e 组成执行数据的算术移位（在 2 的补数系统中的移位）或数据的逻辑移位（无符号移位）。通常，32 位或 64 位数据被输入到桶形移位器 45 或从桶形移位器 45 输出。存储在寄存器 30a 和 30b 中的目标数据的移位量由另一个寄存器指定或根据其立即值指定。对于数据执行左 63 位和右 63 位的范围中的算术或逻辑移位，然后以输入位长度输出。

桶形移位器 45 能够响应于一个 SIMD 指令移动 8-、16-、32-和 64-位数据。例如，桶形移位器 45 可以并行移位四条 8 位数据。

执行在 2 的补数系统中移位的算术移位来用于在加法和减法时对准小数点，以及用于乘以 2 的幂（2，2 的 2nd 次幂，2 的 -1st 次幂）和其他目的。

图 4 是显示变换器 47 的配置的方框图。变换器 47 由饱和块（SAT）47a、BSEQ 块 47b、MSKGEN 块 47c、VSUMB 块 47d、BCNT 块 47e 和 IL 块 47f 组成。

饱和块（SAT）47a 对输入数据执行饱和处理。具有用于 32 位数据的饱和处理的两个块使得能够支持一个对两个数据元素并行执行的 SIMD 指令。

BSEQ 块 47b 从 MSB 开始对连续的 0 或 1 计数。

MSKGEN 块 47c 将一个指定的位段输出为 1, 将其他位输出为 0。

VSUMB 块 47d 将输入数据分成指定位宽度, 并输出其总和。

BCNT 块 47e 对输入数据中指定为 1 的位数计数。

IL 块 47f 将输入数据分成指定位宽度, 并输出从交换每个数据块的的位置产生的值。

图 5 是显示除法器 46 的配置的方框图。使一个被除数是 64 位, 一个除数是 32 位, 除法器 46 输出 32 位分别作为商和模数。涉及 34 个周期来用于获得商和模数。除法器 46 可以处理有符号的和无符号的数据。然而, 应该注意, 对于用作被除数和除数的数据的符号的存在/不存在作出相同的设置。并且, 除法器 46 具有输出一个溢出标志和一个 0 除法标志的能力。

图 6 是显示乘法/乘积和运算单元 44 的配置的方框图。乘法/乘积和运算单元 44 由两个 32 位乘法器 (MUL) 44a 和 44b、三个 64 位加法器 (加法器) 44c~44e、选择器 44f 和饱和处理单元 (饱和) 44g 组成, 执行下面的乘法和乘积和:

32×32 位有符号乘法、乘积和与乘积差;

32×32 位无符号乘法;

在两个数据元素上并行执行的 16×16 位有符号乘法、乘积和与乘积差; 以及

在两个数据元素上并行执行的 32×16 位有符号乘法、乘积和与乘积差;

在以整数和定点格式的数据（h1、h2、w1 和 w2）上执行上面的运算。并且，对这些运算的结果进行四舍五入和饱和。

图 7 是显示指令控制单元 10 的配置的方框图。指令控制单元 10 由指令高速缓存 10a、地址管理单元 10b、指令缓冲器 10c~10e、跳转缓冲器 10f 和旋转单元（旋转）10g 组成，在平常时间和在分支点发出指令。具有三个 128 位指令缓冲器（指令缓冲器 10c~10e）使得能够支持最大数目的并行指令执行。关于分支处理，指令控制单元 10 通过跳转缓冲器 10f 和其他在执行一个分支（settar 指令）之前事先在下面描述的 TAR 寄存器中存储一个分支目的地址。分支采用存储在 TAR 寄存器中的分支目的地址来执行。

应该注意，处理器 1 是一个采用 VLIW 体系结构的处理器。VLIW 体系结构是一种允许多个指令（例如，装入，存储，运算和分支）被存储在一单个指令字中并且这样的指令将在一次全部执行的体系结构。通过由程序员将可以并行执行的一组指令描述为一单个发行组（issue group），这种发行组可以被并行执行。在这个说明书中，发行组的分隔符由“;;”指示。下面描述了符号例子。

（例子 1）

```
mov r1, 0x23;;
```

这个指令描述指示只有一个指令“mov”应该被执行。

（例子 2）

```
mov r1, 0x38
```

```
add r0, r1, r2
```

```
sub r3, r1, r2;;
```

这些指令描述指示三个指令“mov”、“add”和“sub”应该被并行执行。

指令控制单元 10 识别一个发行组并将其发送到译码单元 20。译码单元 20 对发行组中的指令译码，并控制用于执行这样的指令所需的资源。

接着，对于在处理器 1 中包括的寄存器给出解释。

下面的表 1 列出了处理器 1 的一组寄存器。

[表 1]

寄存器名	位宽度	寄存器数	用法
R0~R31	32 位	32	通用寄存器。在执行运算指令时用作数据存储器指针、数据存储器等。
TAR	32 位	1	分支寄存器。用作在分支点的分支地址存储器。
LR	32 位	1	连接寄存器。
SVR	16 位	2	保存寄存器。用于保存条件标志（CFR）和各种模式。
M0~M1 (MH0:ML0 ~MH1~ML 1)	64 位	2	运算寄存器。在执行运算指令时用作数据存储器。

下面的表 2 列出了处理器 1 的一组标志（在后面描述的条件标志寄存器等中管理的标志）。

[表 2]

标志名	位宽度	标志数	用法
C0~C7	1	8	条件标志。指示条件是否建立。
VC0~VC3	1	4	用于媒体处理扩充指令的条件标志。指示条件是否建立。
OVS	1	1	溢出标志。在运算时检测溢出。
CAS	1	1	进位标志。在运算时检测进位。
BPO	5	1	指定位位置。指定在执行屏蔽处理指令时要处理的位位置。
ALN	2	1	指定的字节对准。
FXP	1	1	定点运算模式。
UDR	32	1	未定义的寄存器。

图 8 是显示通用寄存器（R0~R31）30a 的配置的示意图。通用寄存器（R0~R31）30a 是构成一个要执行的任务的上下文的主要部分并存储数据或地址的一组 32 位寄存器。应该注意，通用寄存器 R30 和 R31 分别通过硬件用作全局指针和堆栈指针。

图 9 是显示连接寄存器（LR）30c 的配置的示意图。与这个连接寄存器（LR）30c 相连，处理器 1 还具有一个在图中未显示的保存寄存器（SVR）。连接寄存器（LR）30c 是一个用于在函数调用时存储返回地址的 32 位寄存器。应该注意，保存寄存器（SVR）是一

个用于在函数调用时保存条件标志寄存器的条件标志（CFR.CF）的 16 位寄存器。与在后面将解释的分支寄存器（TAR）的情况中一样，连接寄存器（LR）30c 还被用于提高循环速度的目的。作为较低的 1 位总是读出 0，但在写操作时必须写入 0。

例如，在执行“调用”（brr, jmprr）指令时，处理器 1 将一个返回地址保存在连接寄存器（LR）30c 中，并将一个条件标志（CFR.CF）保存在保存寄存器（SVR）中。在执行“jmp”指令时，处理器 1 从连接寄存器（LR）30c 取出返回地址（分支目的地址），并返回一个程序计数器（PC）。此外，在执行“ret(jmrr)”指令时，处理器 1 从连接寄存器（LR）30c 取出分支目的地址（返回地址），并将其存储到程序计数器（PC）。此外，处理器 1 从保存寄存器（SVR）取出条件标志，以便将其存储（再存入）到条件标志寄存器（CFR）32 中的条件标志区域 CFR.CF。

图 10 是显示分支寄存器（TAR）30d 的配置的示意图。分支寄存器（TAR）30d 是一个用于存储分支目标地址的 32 位寄存器，主要用于提高循环速度的目的。作为较低的 1 位总是读出 0，但在写操作时必须写入 0。

例如，当执行“jmp”和“jloop”指令时，处理器 1 从分支寄存器（TAR）30d 取出一个分支目的地址，并将其存储在程序计数器（PC）中。当由存储在分支寄存器（TAR）30d 中的地址所指示的指令被存储在分支指令缓冲器中时，分支损失（penalty）将为 0。通过将一个循环的顶端地址存储在分支寄存器（TAR）30d 中，可

以实现循环速度的提高。

图 11 是显示程序状态寄存器 (PSR) 31 的配置的示意图。程序状态寄存器 (PSR) 31 构成一个要执行的任务的上下文的主要部分，是一个用于存储下面的处理器状态信息的 32 位寄存器：

位 SWE：指示 VMP（虚拟多处理器）到 LP（逻辑处理器）的切换是被允许还是禁止。“0”指示到 LP 的切换被禁止，“1”指示到 LP 的切换被允许。

位 FXP：指示一个定点模式。“0”指示模式 0（在其中在假设在 MSB 和从 MSB 开始的第一位之间有小数点的情况下执行一个算术运算的模式。此后，也称为“_1 系统”），“1”指示模式 1（在其中在假设在从 MSB 开始的第一位和从 MSB 开始的第二位之间有小数点的情况下执行一个算术运算的模式。此后，也称为“_2 系统”）。

位 IH：是一个中断处理标志，指示可屏蔽的中断处理是否是正在进行的。“1”指示有一个正在进行的中断处理，“0”指示没有正在进行的中断处理。在一个中断出现时这个标志被自动设置。这个标志用于对在程序中的处理器响应于“rti”指令而返回的一点处正在进行中断处理还是程序处理作出区分。

位 EH：是一个指示错误或 NMI 是否正在被处理的标志。“0”指示错误/NMI 中断处理不是正在进行的，“1”指示错误/NMI 中断处理是正在进行的。如果在 EH=1 时出现不同步错误或 NMI，这个标志被屏蔽。同时，当 VMP 被允许时，VMP 的板切换被屏蔽。

位 PL [1:0]：指示一个特权级。“00”指示特权级 0，即处理器抽

象级，“01”指示特权级 1（不可设置），“10”指示特权级 2，即系统程序级，“11”指示特权级 3，即用户程序级。

位 LPIE3：指示 LP 专用中断 3 是被允许还是禁止。“1”指示一个中断被允许，“0”指示一个中断被禁止。

位 LPIE2：指示 LP 专用中断 2 是被允许还是禁止。“1”指示一个中断被允许，“0”指示一个中断被禁止。

位 LPIE1：指示 LP 专用中断 1 是被允许还是禁止。“1”指示一个中断被允许，“0”指示一个中断被禁止。

位 LPIE0：指示 LP 专用中断 0 是被允许还是禁止。“1”指示一个中断被允许，“0”指示一个中断被禁止。

位 AEE：指示一个未对准异常是被允许还是禁止。“1”指示一个未对准异常被允许，“0”指示一个未对准异常被禁止。

位 IE：指示一个级别中断是被允许还是禁止。“1”指示一个级别中断被允许，“0”指示一个电平中断被禁止。

位 IM [7:0]：指示一个中断屏蔽，并且范围是从级 0~7，每个都能够在其自己的级别被屏蔽。级 0 是最高级。在未由任何 IM 屏蔽的中断请求中，只有具有最高级别的中断请求被处理器 1 接受。当一个中断请求被接受时，在所接受的级别之下的级别被硬件自动屏蔽。IM[0]代表级 0 的屏蔽，IM[1]代表级 1 的屏蔽，IM[2]代表级 2 的屏蔽，IM[3]代表级 3 的屏蔽，IM[4]代表级 4 的屏蔽，IM[5]代表级 5 的屏蔽，IM[6]代表级 6 的屏蔽，IM[7]代表级 7 的屏蔽。

保留：指示一个保留位。总是读出 0。在写入时必须写 0。

图 12 是显示条件标志寄存器（CFR）32 的配置的示意图。条件标志寄存器（CFR）32 构成一个要执行的任务的上下文的主要部分，是一个由条件标志、运算标志、向量条件标志、运算指令位位置说明字段和 SIMD 数据对准信息字段组成的 32 位寄存器。

位 ALN [1:0]：指示一个对准模式。设置“valnvc”指令的对准模式。

位 BPO [4:0]：指示一个位位置。它用在需要位位置说明的指令中。

位 VC0~VC3：是向量条件标志。从 LSB 侧上的一个字节或半个字开始到 MSB 侧，每个对应于从 VC0 到 VC3 的范围内的一个标志。

位 OVS：是一个溢出标志（总计）。它是在饱和和溢出的检测基础上设置的。如果未检测到，保持在执行指令之前的一个值。这个标志的清除需要由软件实现。

位 CAS：是一个进位标志（总计）。它当在“addc”指令下出现一个进位时或当在“subc”指令下出现一个借位时被设置。如果在“addc”指令下未出现进位时或在“subc”指令下未出现借位，保持在执行指令之前的一个值。这个标志的清除需要由软件实现。

位 C0~C7：是条件标志，指示一个带有条件的执行指令中的条件（TRUE/FALSE）。带有条件的执行指令的条件和位 C0~C7 之间的对应关系由指令中包括的判定位来决定。应该注意，标志 C7 的值总是 1。对标志 C7 作出的 FALSE 条件的反映（写入 0）被忽略。

保留：指示一个保留位。总是读出 0。在写入时必须写入 0。

图 13A 和 13B 是显示累加器 (M0, M1) 30b 的配置的示意图。这种累加器 (M0, M1) 30b 构成一个要执行的任务的上下文的主要部分, 由图 13A 所示的 32 位寄存器 MHO-MH1 (用于乘法和除法/乘积和 (高 32 位) 的寄存器) 和图 13B 所示的 32 位寄存器 MLO-ML1 (用于乘法和除法/乘积和 (低 32 位) 的寄存器) 组成。

寄存器 MHO-MH1 在一个乘法指令时用于存储运算结果的高 32 位, 而在一个乘积和指令时用作累加器的高 32 位。此外, 在处理一个位流的情况下, 寄存器 MHO-MH1 可以与通用寄存器结合使用。同时, 寄存器 MLO-ML1 在一个乘法指令时用于存储运算结果的低 32 位, 而在一个乘积和指令时用作累加器的低 32 位。

图 14 是显示程序计数器 (PC) 33 的配置的示意图。这个程序计数器 (PC) 33 构成一个要执行的任务的上下文的主要部分, 是一个保存要执行的指令的地址的 32 位计数器。

图 15 是显示 PC 保存寄存器 (IPC) 34 的配置的示意图。这个 PC 保存寄存器 (IPC) 34 构成一个要执行的任务的上下文的主要部分, 是一个 32 位寄存器。

图 16 是显示 PSR 保存寄存器 (IPSR) 35 的配置的示意图。这个 PSR 保存寄存器 (IPSR) 35 构成一个要执行的任务的上下文的主要部分, 是一个用于保存程序状态寄存器 (PSR) 31 的 32 位寄存器。作为对应于一个保留位的一部分总是读出 0, 但在写入时必须写入 0。

接着, 对于处理器 1 的存储空间给出解释, 这是依据本实施例

的编译程序的目标。例如，在处理器 1 中，带有 4GB 容量的线性存储空间被分成 32 段，并将一个指令 SRAM（静态 RAM）和一个数据 SRAM 分配给 128MB 段。采用一个 128MB 段用作一个块，在 SAR（SRAM 区域寄存器）中设置要访问的目标块。当所访问的地址是 SAR 中设置的段时，对指令 SRAM/数据 SRAM 作出直接访问，但当这样的地址不是在 SAR 中设置的段时，访问请求应该被发到总线控制器（BCU）。片上存储器（OCM）、外部存储器、外部设备、I/O 端口等连接到 BUC。从这些设备的数据读取和向这些设备的数据写入是可能的。

图 17 是显示处理器 1 的流水线行为的时序图，这是依据本实施例的编译程序的目标。如图所示，处理器 1 的流水线基本上包括下面五个阶段：指令取出；指令分配（调度）；译码；执行和写入。

图 18 是显示在执行一个指令时的处理器 1 的流水线行为的每个阶段的时序图。在指令取出阶段，对由程序计数器（PC）33 所指定的地址所指示的指令存储器作出访问，并将指令传送到指令缓冲器 10c~10e 等。在指令分配阶段，实现响应于分支指令的分支目的地址信息的输出、输入寄存器控制信号的输出、可变长度指令的分配，随后将指令传送到指令寄存器（IR）。在译码阶段，将 IR 输入到译码单元 20，并输出一个运算单元控制信号和一个存储器访问信号。在执行阶段，执行一个运算，运算的结果被输出到数据存储器或通用寄存器（R0~R31）30a。在写入阶段，将作为数据传送的结果而获得的值以及运算结果存储在通用寄存器中。

作为依据本实施例的编译程序的目标的处理器 1 的 VLIW 结构允许在最多三个数据元素上并行执行上面的处理。因此，处理器 1 以图 19 所示的定时并行执行图 18 所示的行为。

接着，对于由具有上述配置的处理器 1 执行的一组指令给出解释。

表 3~5 列出了要由作为依据本实施例的编译程序的目标的处理器 1 执行的分类的指令。

[表 3]

类别	运算单元	指令运算代码
存储器传送指令 (装入)	M	ld, ldh, ldhu, ldb, ldbu, ldp, ldhp, ldbp, ldbh, ldbuh, ldbhp, ldbuhp
存储器传送指令 (存储)	M	st, sth, stb, stp, sthp, stbp, stbh, stbhp
存储器传送指令 (其它)	M	dpref, ldstb
外部寄存器传送指令	M	rd, rde, wt, wte
分支指令	B	br, brl, call, jmp, jmpl, jmp, ret, jmpf, jloop, setbb, setlr, settar
软件中断指令	B	rti, pi0, pi0l, pi1, pi1l, pi2, pi2l, pi3, pi3l, pi4, pi4l, pi5, pi5l, pi6, pi6l, pi7, pi7l, sc0, sc1, sc2, sc3, sc4, sc5, sc6, sc7

类别	运算单元	指令运算代码
VMP/中断控制指令	B	intd, inte, vmpsleep, vmpsus, vmpswd, vmpswe, vmpwait
算术运算指令	A	abs, absvh, absvw, add, addarvw, addc, addmsk, adds, addsr, addu, addvh, addvw, neg, negvh, negvw, rsub, sladd, s2add, sub, subc, submsk, subs, subvh, subvw, max, min
逻辑运算指令	A	and, andn, or, sethi, xor, not
比较指令	A	cmpCC, cmpCCa, cmpCCn, cmpCCo, tstn, tstna, tstnn, tstno, tstz, tstza, tstzn, tstzo
移动指令	A	mov, movcf, mvclcas, mvclovs, setlo, vcchk
NOP 指令	A	nop
移位指令 1	S1	asl, aslvh, aslvw, asr, asrvh, asrvw, lsl, lsr, rol, ror
移位指令 2	S2	aslp, aslpvw, asrp, asrpvw, lslp, lsrp

[表 4]

类别	运算单元	指令运算代码
提取指令	S2	ext, extb, extbu, exth, exthu, extr, extru, extu
屏蔽指令	C	msk, mskgen
饱和指令	C	sat12, sat9, satb, satbu, sath, satw

类别	运算单元	指令运算代码
变换指令	C	valn, valn1, valn2, valn3, valnvc1, valnvc2, valnvc3, valnvc4, vhpkb, vhpkh, vhunpkb, bhunpkh, vintlhb, vintlhh, bintlhb, vintlhh, vlpkb, vlpkbu, vlpkh, vlpkhu, vlunpkb, vlunpkbu, vlunpkh, vlunpkhu, vstovb, vstovh, vunpk1, vunpk2, vxchngh, vexth
位计数指令	C	bcnt1, bseq, bseq0, bseq1
其它	C	byterev, extw, mskbrvb, mskbrvh, rndvh, movp
乘法指令 1	X1	fmulhh, fmulhhr, fmulhw, fmulhww, hmul, lmul
乘法指令 2	X2	fmulww, mul, mulu
乘积和指令 1	X1	fmachh, fmachhr, fmachw, fmachww, hmac, lmac
乘积和指令 2	X2	fmacww, mac
乘积差指令 1	X1	fmsuhh, fmsuhhr, fmsuhw, fmsuww, hmsu, lmsu
乘积差指令 2	X2	fmsuww, msu
除法指令	DIV	div, divu
调试程序指令	DBGM	dbgm0, dbgml, dbgml2, dbgml3

[表 5]

类别	运算单元	指令运算代码
SIMD 算术运算指令	A	vabshvh, vaddb, vaddh, vaddhvc, vaddhvh, vaddrhvc, vaddsb, vaddsh, vaddsrh, vaddsrh, vasubb, vcchk, vhaddh, vhaddhvh, vhsbh, vhsbhvh, vladdh, vladdhvh, vlsubh, vlsubhvh, vnegb, vnegh, vneghvh, vsaddb, vsaddh, vsgrh, vsrsubb, vsrsubh, vssubb, vssubh, vsubb, vsubh, vsubhvh, vsubsh, vsumh, vsumh2, vsumrh2, vxaddh, vxaddhvh, vxsubh, vxsubhvh, vmaxb, vmaxh, vminb, vminh, vmovt, vsel
SIMD 比较指令	A	vcmpeqb, vcmpeqh, vcmpgeb, vcmpgeh, vcmpgtb, vcmpgth, vcmpleb, vcmpleh, vcmpltb, vcmplth, vcmpneb, vcmpneh, vscmpeqb, vscmpeqh, vscmpgeb, vscmpgeh, vscmpgtb, vscmpgth, vscmpleb, vscmpleh, vscmpltb, vscmplth, vscmpneb, vscmpneh
SIMD 移位指令 1	S1	vaslb, vaslh, vaslvh, vasrb, vasrh, vasrvh, vlslb, vlslh, vlslrb, vlslrh, vrohb, vrohl, vrohb, vrohr
SIMD 移位指令 2	S2	vasl, vaslvw, vasr, vasrvw, vlsl, vlslr
SIMD 饱和指令	C	vsath, vsath12, vsath8, vsath8u, vsath9
其它 SIMD 指令	C	vabssumb, vrndvh
SIMD 乘法指令	X2	vfmulh, vfmulhr, vfmulw, vhfmulh,

类别	运算单元	指令运算代码
		vhfmulhr, vhfmulw, vhmul, vlfmulh, vlfmulhr, vlfmulw, vlmul, vmul, vpfmulhww, vxfmulh, vxfmulhr, vxfmulw, vxmul
SIMD 乘积和指令	X2	vfmach, vfmachr, vfmacw, vhfmach, vhfmachr, vhfmacw, vhmach, vlfmach, vlfmachr, vlfmacw, vlmach, vmach, vpfmachww, vxfmach, vxfmachr, vxfmachw, vxmach
SIMD 乘积差指令	X2	vfmsuh, vfmsuw, vhfmsuh, vhfmsuw, vhmsu, vlfmsuh, vlfmsuw, vlmsu, vmsu, vxfmsuh, vsfmsuw, vxmsu

应该注意，上面的表中的“运算单元”指的是用在各个指令中的运算单元。更具体地，“A”代表 ALU 指令，“B”代表分支指令，“C”代表变换指令，“DIV”代表除法指令，“DBGM”代表调试指令，“M”代表存储器访问指令，“S1”和“S2”代表移位指令，“X1”和“X2”代表乘法指令。

图 20 是显示由处理器 1 执行的指令的格式的示意图。

下面描述图中的首字母缩写词代表什么含义：“P”是判定（执行条件：指定八个条件标志 C0~C7 中的一个）；“OP”是运算代码字段；“R”是寄存器字段；“I”是立即字段；“D”是位移字段。

图 21~36 是解释由处理器 1 执行的指令的概略功能的示意图。

更具体地，图 21 解释一个属于类别“ALUadd（加法）系统”的指令；图 22 解释一个属于类别“ALUsub（减法）系统”的指令；图 23 解释一个属于类别“ALUlogic（逻辑运算）系统等”的指令；图 24 解释一个属于类别“CMP（比较运算）系统”的指令；图 25 解释一个属于类别“mul（乘法）系统”的指令；图 26 解释一个属于类别“mac（乘积和运算）系统”的指令；图 27 解释一个属于类别“msu（乘积差）系统”的指令；图 28 解释一个属于类别“MEMld（从存储器装入）系统”的指令；图 29 解释一个属于类别“MEMstore（存储在存储器中）系统”的指令；图 30 解释一个属于类别“BRA（分支）系统”的指令；图 31 解释一个属于类别“BSasl（算术桶形移位）系统等”的指令；图 32 解释一个属于类别“BSasl（逻辑桶形移位）系统等”的指令；图 33 解释一个属于类别“CNVvaln（算术变换）系统”的指令；图 34 解释一个属于类别“CNV（普通变换）系统”的指令；图 35 解释一个属于类别“SATvlpk（饱和处理）系统”的指令；图 36 解释一个属于类别“ETC（等等）系统”的指令。

图 20 是显示由处理器 1 执行的指令的格式的示意图。

下面描述图中的首字母缩写词代表什么含义：“P”是判定（执行条件：指定八个条件标志 C0~C7 中的一个）；“OP”是运算代码字段；“R”是寄存器字段；“I”是立即字段；“D”是位移字段。

图 21~36 是解释由处理器 1 执行的指令的概略功能的示意图。更具体地，图 21 解释一个属于类别“ALUadd（加法）系统”的指令；图 22 解释一个属于类别“ALUsub（减法）系统”的指令；图

23 解释一个属于类别“ALUlogic（逻辑运算）系统等”的指令；图 24 解释一个属于类别“CMP（比较运算）系统”的指令；图 25 解释一个属于类别“mul（乘法）系统”的指令；图 26 解释一个属于类别“mac（乘积和运算）系统”的指令；图 27 解释一个属于类别“msu（乘积差）系统”的指令；图 28 解释一个属于类别“MEMld（从存储器装入）系统”的指令；图 29 解释一个属于类别“MEMstore（存储在存储器中）系统”的指令；图 30 解释一个属于类别“BRA（分支）系统”的指令；图 31 解释一个属于类别“BSasl（算术桶形移位）系统等”的指令；图 32 解释一个属于类别“BSasl（逻辑桶形移位）系统等”的指令；图 33 解释一个属于类别“CNVvaln（算术变换）系统”的指令；图 34 解释一个属于类别“CNV（普通变换）系统”的指令；图 35 解释一个属于类别“SATvlpk（饱和处理）系统”的指令；图 36 解释一个属于类别“ETC（等等）系统”的指令。

下面描述这些图中的每列的含义：“SIMD”指示一个指令的类型（在 SISD（SINGLE）和 SIMD 之间区分）；“大小”指示要作为运算目标的单个运算数的大小；“指令”指示一个运算的运算代码；“运算数”指示一个指令的运算数；“CFR”指示条件标志寄存器中的改变；“PSR”指示处理器状态寄存器中的改变；“典型行为”指示一个行为的概述；“运算单元”指示一个要使用的运算单元；以及，“3116”指示一个指令的大小。

下面解释涉及用在后面将描述的具体例子中的主要指令的处理器 1 的行为。

andn Rc,Ra,Rb

实现 Ra 和 Rb 之间的反转逻辑 AND 并将其存储在 Rc 中。

asl Rb,Ra,I5

执行向 Ra 左移立即值 (I5) 中的位数的算术移位。

and Rb,Ra,I8

实现 Ra 和值 (I8) 之间的逻辑 AND 并将其存储在 Rb 中。

bseq0 Rb,Ra

对从 Ra 的 MSB 开始的连续的 0 计数并将其存储在 Rb 中。

bseq1 Rb,Ra

对从 Ra 的 MSB 开始的连续的 1 计数并将其存储在 Rb 中。

bseq Rb,Ra

对从 Ra 的 MSB 之下 1 位开始的连续的符号位计数并将其存储在 Rb 中。当 Ra 是 0 时，输出 0。

bcnt1 Rb,Ra

对 Ra 的 1 的个数计数并将其存储在 Rb 中。

extr Rc,Ra,Rb

由 Rb 指定一个位的位置，提取 Ra 的内容一部分，符号扩展并将其存储在 Rc 中。

extru Rc,Ra,Rb

由 Rb 指定一个位的位置，提取 Ra 的内容一部分，不带符号扩展地将其存储在 Rc 中。

fmulhh Mm,Rc,Ra,Rb

将 Ra、Rb 和 Rc 作为 16 位值来处理，将 Mm（用于乘法的累加数）作为 32 位值来处理。将 Ra 和 Rb 与一定点相乘。将结果存储在 Mm 和 Rc 中。当结果不能由有符号的 32 位表示时，使其饱和。

`fmulhw Mm,Rc,Ra,Rb`

将 Ra 和 Rb 作为 16 位值来处理，将 Mm 和 Rc 作为 32 位值来处理。通过一定点将 Ra 和 Rb 相乘。将结果存储在 Mm 和 Rc 中。当结果不能由有符号的 32 位表示时，使其饱和。

`mul Mm,Rc,Ra,Rb`

将 Ra 和 Rb 与一整数相乘。将结果存储在 Mm 和 Rc 中。

`mac Mm,Rc,Ra,Rb,Mn`

将 Ra 和 Rb 与一整数相乘并将其加到 Mn。将结果存储在 Mm 和 Rc 中。

`mov Rb,Ra`

将 Ra 传送到 Rb。

`or Rc,Ra,Rb`

实现 Ra 和 Rb 之间的逻辑 OR 并将其存储在 Rc 中。

`rde C0:C1,Rb,(Ra)`

让 Ra 是一个外部寄存器数，将外部寄存器的值读入 Rb。将读取的成功和失败分别输出到 C0 和 C1（条件标志）。在失败的情况下，出现一个扩展寄存器错误的异常。

`wte C0:C1,(Ra),Rb`

让 Ra 是一个外部寄存器数，将 Rb 的值写入外部寄存器。将写

入的成功和失败分别输出到 C0 和 C1。在失败的情况下，出现一个扩展寄存器错误的异常。

vaddh Rc,Ra,Rb

以半个字向量格式对待每个寄存器。将 Ra 和 Rb 相加（SIMD 直接）。

（一个编译程序）

接着，解释依据本实施例的、其目标是上述处理器 1 的一个编译程序。

图 37 是显示依据本实施例的编译程序 100 的配置的功能方框图。这个编译程序 100 是一个将以诸如 C/C++语言的高级程序语言描述的源程序 101 翻译成其目标处理器是上述处理器 1 的机器语言程序 105 的交叉编译程序，由一个在诸如个人计算机的计算机上执行的程序实现，并主要被分成并带有分析程序单元 110、中间编码变换单元 120、优化单元 130 和代码产生单元 140。

应该注意，有效地产生专用于上面提到的处理器 1 的专用指令的主文件（运算符定义文件 102 和内置函数定义文件 103）在当前编译程序 100 中是准备好的。用户可以通过将这些主文件包括在源程序 101 中来获取为处理器 1 专门做（优化）的机器语言程序 105。

如图 38 到图 68 中的清单例子所示，运算符定义文件 102 是一个定义了定义以定点和 SIMD 类型数据为目标的运算符的类的主文件。在该主文件中，图 38~图 40 是定义了一个其目标主要是模式 0

(_1 系统) 的 16 位定点的数据的运算符的部分的清单; 图 41 和图 42 是定义了一个其目标主要是模式 0 (_1 系统) 的 32 位定点的数据的运算符的部分的清单; 图 43~图 45 是定义了一个其目标主要是模式 1 (_2 系统) 的 16 位定点的数据的运算符的部分的清单; 图 45~图 47 是其目标主要是模式 1 (_2 系统) 的 32 位定点的数据的运算符的部分的清单; 图 48~图 68 是定义了其他函数的部分的清单。

如图 69~图 72 中的清单例子所示, 内置函数定义文件 103 是一个定义了执行各种运算来以专用于处理器 1 的机器语言指令代替函数的函数的主文件。在该主文件中, 图 69~图 71 是定义了一个以一个机器语言指令代替函数的函数的部分的清单; 图 72 是定义了一个以两个或更多机器语言指令(机器语言指令序列)代替函数的函数的部分的清单。

应该注意, 这些定义文件 102 和 103 中的 `asm(...){...}(...)` 是一个被称为优化 `asm` 的内置汇编程序指令, 其处理如下。换句话说, 优化 `asm` 语句的描述格式是

```
asm(<<装入表达式的清单>>){  
    <<优化控制信息>>  
    <<指令指定单元>>  
}(<<存储表达式的清单>>);。
```

这里, “装入表达式的清单” 是一个描述装入表达式的部分; “装入表达式” 是一个存储 C 语言中的变量和诸如四个运算的表达式的结果的表达式; 它被描述为类似 “寄存器指定标识符=赋值表达式”;

以及，它意味着在右边指示的值被传送到在左边指示的标识符。“存储表达式的清单”是一个描述存储表达式的部分；“存储表达式”被描述为类似“单项式=寄存器指定标识符”；以及，它意味着将左边的由单项式表示的值赋给由寄存器指定标识符表示的寄存器的值。

分析程序单元 110 是一个提取保留字（关键字）等的前处理单元；实现作为编译的目标的（包含要包括的主文件的）源程序 101 的词汇分析；以及，除了普通编译程序具有的分析函数之外，还具有一个支持在定点上的模式的切换的定点模式切换单元 111。当定点模式切换单元 111 在源程序 101 中检测到一个保存和恢复定点模式的编译指示说明（例如，“`pragma_save_fxpmode func`”）时，它产生一个保存和恢复处理器 1 的 PSR31 的位 FXP 的机器语言指令。这实现了定点的模式 0 和模式 1 中的运算混合的程序设计。

应该注意，“编译指示（或编译指示说明）”是用户可以在源程序 101 中任意指定（放置）的对编译程序 100 的指示，是以“`#pragma`”开始的字符序列。

中间编码变换单元 120 是一个将从分析程序单元 110 传送的源程序 101 中的每个语句变换到中间编码的处理单元，由中间编码产生单元 121 和机器语言指令替代单元 122 组成。中间编码产生单元 121 根据一个预定规则变换源程序 101 中的每个语句。这里，中间编码典型地是以函数调用的格式表示的代码（例如，指示“`+(int a, int b)`”；指示“将一个整数 a 加到整数 b”的代码）。但中间编码不仅包含具有函数调用格式的代码，还包含处理器 1 的机器语言指令。

机器语言指令替代单元 122 将由中间编码产生单元 121 产生的中间编码中的具有函数调用格式的中间编码变换成引用运算符定义文件 102 和内置函数定义文件 103 的对应的机器语言指令（或机器语言指令序列），将匹配由这些定义文件或内置函数定义的运算符（包括运算的目标数据的类型）的中间编码变换成遵循（follow）机器语言指令替代单元 122 在其自身内部具有的一个替代表 122a 或由这些定义文件定义的汇编程序指令变换成对应的机器语言指令（或机器语言指令序列），并将变换的机器语言指令输出到优化单元 130。这允许优化单元 130 对这些中间编码执行各种优化，因为它们不是以内置函数的格式而是以机器语言指令的格式传到优化单元 130。

附带地，替代表 122a 是一个存储对应于事先保留的运算符的运算和函数的机器语言指令（机器语言指令序列）的表。另外，机器语言指令替代单元 22 输出来自从中间编码产生单元 121 传送的中间编码的机器语言指令而不经优化单元 130。

优化单元 130 是一个处理单元，通过执行诸如合并指令、去除冗余、分类指令和分配寄存器的处理，对来自从中间编码变换单元 120 输出的中间编码的机器语言指令执行由用户选择的下面提到的三种类型的优化中的一种：（1）提高执行速度具有更高优先级的优化；（2）减小代码大小具有更高优先级的优化；以及（3）提高执行速度和减小代码大小的优化。优化单元 130 具有一个除了普通优化之外还执行对本编译程序 100 的特有优化（例如“循环展开”、“如果变换”和“成对存储器访问指令的产生”）的处理单元（变元优化

单元 131、类型变换优化单元 132 和等待时间优化单元 133)。

变元优化单元 131 是一个根据内置函数（例如，`extr`，`extru`）的变元产生适当的指令或序列（算法）的处理单元。例如，当所有变元是常数时，变元优化单元 131 产生其运算数是通过保持在常数而获得的常数值的机器语言指令；当变元的一部分是常数时，产生其运算数是立即值的机器语言指令；当所有变元是变量时，产生其运算数是寄存器的一个指令序列。

类型变换优化单元 132 是一个基于在源程序 101 中的一个特定记号使得在不同类型之间的运算更有效的处理单元。例如，当希望一个 16 位数据和另一个 16 位数据的乘法结果被保持为一个 32 位数据时，如果在源程序 101 中有特定记号，类型变换优化单元 132 产生一个以这样一种类型的变换执行乘法的机器语言指令（“`fmulhw`”等）。

等待时间优化单元 133 基于在源程序 101 中结合的一个汇编程序指令中的关于等待时间的指示（周期数的指定），对准机器语言指令，以便一特定部分或一特定行动只花费指定周期数的执行时间。这使得一个程序员不必要完成他插入所需数目的“`nop`（空操作）”指令的常规工作，并使得能够通过插入其他机器语言指令而不是“`nop`”指令来执行优化。

附带地，“循环展开”是通过扩展循环的迭代（重复）并产生一对存储器访问指令（`ldp/stp/ldhp/sthp` 等）以便同时执行多个迭代来提高并行执行循环的可能性的优化。另外，“如果转换”是通过为一

个带有条件的执行机制产生一个指令（只有在指令中包括的条件（判定）匹配处理器 1 的状态（条件标志）时才执行的指令）来去除分支结构的优化。此外，“一对存储器访问指令的产生”是以一对寄存器（两个连续的寄存器）作为目标产生所述一对存储器访问指令（ldp/stp/ldhp/sthp 等）的优化。

此外，优化单元 130 输出来自函数调用格式的中间编码的、不能被扩展的中间编码，而不经过程序产生单元 140，因为不可能在上述机器语言指令级执行优化处理。

代码产生单元 140 参考内部保存的翻译表等产生机器语言程序 105，代替从优化单元 130 输出的所有中间编码（包括函数调用格式的代码和优化的机器语言指令）。

接着，解释指示具体例子的如上所述配置的编译程序 100 的特征行为。

图 73 是显示机器语言指令替代单元 122 的行为的流程图。机器语言指令替代单元 122 重复下面的过程：（1）判断来自中间编码产生单元 121 产生的中间编码的函数调用格式的代码是否匹配由运算符定义文件 102 定义的运算符（包括运算目标的数据类型）和由内置函数定义文件 103 定义的函数（步骤 S102），以及，当它们匹配时（在步骤 S102 为是），（2）遵循由机器语言指令替代单元 122a 在其自身内部具有的替代表 122a 定义的汇编程序指令和这些定义文件 102 和 103（步骤 S100~S103），用机器语言指令替代运算符和函数（步骤 S102）。

更具体地，不同类型之间的类型变换的隐含规则等由运算符定义文件 102 的定义（由构造符的定义）来规定；定义了下面四种类型的定点：

“FIX16_1”：有符号的 16 位，小数点在第 14 位和第 15 位(MSB)之间，

“FIX16_2”：有符号的 16 位，小数点在第 13 位和第 14 位之间，

“FIX32_1”：有符号的 32 位，小数点在第 30 位和第 31 位(MSB)之间，

“FIX32_2”：有符号的 32 位，小数点在第 29 位和第 30 位之间。

因此，机器语言指令替代单元 122 例如将一个源程序

```
FIX16_1 a,b,c;
```

```
c=a*b;
```

用一个机器语言指令

`fmulhh m0,Rc,Ra,Rb`（定点乘法运算指令）来替代。

因此，用户可以声明四种类型 `FIX16_1`、`FIX16_2`、`FIX32_1` 和 `FIX32_2` 与一个普通编译程序的标准类型类似并使用它们。然后，产生的包括相邻代码的机器语言指令变为优化单元 130 中的诸如合并指令、去除冗余、分类指令和分配寄存器的优化的目标，并且可以被优化。

类似地，不同类型之间的类型转换的隐含规则等由运算符定义文件 102 的定义（由构造符的定义）来规定；定义了下面四种类型的 SIMD 指令：

“VINT8×4”；4 个并行的 8 位整数数据，

“VINT16×2”；2 个并行的 16 位整数数据，

“VFIX161×2”；2 个并行的模式 0（_1 系统）的 16 位定点数据，以及

“VFIX162×2”；2 个并行的模式 1（_2 系统）的 16 位定点数据。因此，机器语言指令替代单元 122 例如将一个源程序

```
VINT16×2 a,b,c;
```

```
c=a+b;
```

用一个机器语言指令

vaddh Rc,Ra,Rb（SIMD 加法指令）替代。

从而，用户可以声明四种类型“VINT8×4”、“VINT16×2”、“VFIX161×2”和“VFIX162×2”与一个普通编译程序的标准类型类似并使用它们。然后，产生的包括相邻代码的机器语言指令变为优化单元 130 中的诸如合并指令、去除冗余、分类指令和分配寄存器的优化的目标，并且可以被优化。

另外，在内置函数定义文件 103 中，定义一个可以使用处理器 1 执行的高级指令的函数（例如，“_abs(a)”等）及其对应的高级指令（例如，一个机器语言指令“abs Rb,Ra”等）。因此，机器语言指令替代单元 122 例如将一个源程序

```
b=_abs(a);
```

用一个机器语言指令

abs Rb,Ra 来替代。

从而，用户可以通过不以 C++语言和汇编程序指令生成而是仅仅调用一个实现准备好的内置函数来实现一个复杂处理。然后，产生的包括相邻代码的机器语言指令变为优化单元 130 中的诸如合并指令、去除冗余、分类指令和分配寄存器的优化的目标，并且可以被优化。

类似地，在内置函数定义文件 103 中，定义一个可以使用处理器 1 执行的高级指令的函数（例如，“_div(a,b)”等）及其对应的高级指令（例如，一个机器语言指令序列“extw, aslp, div”等）。因此，机器语言指令替代单元 122 例如将一个源程序

```
c=_div(a,b);
```

用一个机器语言指令序列

```
extw Mn,Rc,Ra
```

```
aslp Mn,Rc, Mn,Rc,15
```

```
div MHm, Rc, MHn,Rc,Rb.
```

来替代。

从而，用户可以通过不以 C++语言和汇编程序指令生成而是仅仅调用一个实现准备好的内置函数来实现一个复杂处理。然后，产生的包括相邻代码的机器语言指令变为优化单元 130 中的诸如合并指令、去除冗余、分类指令和分配寄存器的优化的目标，并且可以被优化。

应该注意，在内置函数定义文件 103 中列出的内置函数中，（1）被变换成一个机器语言指令的函数，（2）被变换成两个或更多机器

语言指令（一个机器语言指令序列）的函数，以及（3）可以指定不是寄存器分配的目标的资源（例如累加器）的函数的代表性例子如下：

（1）被变换成一个机器语言指令的内置函数

“_bseq1(x)”：

这是检测从输入的 MSB 开始有多少个连续的位 0 的函数。其格式如下：

```
int_bseq1(FIX16_1 val) // count 1
```

```
int_bseq1(FIX16_2 val) // count 1
```

```
int_bseq1(FIX32_1 val) // count 1
```

```
int_bseq1(FIX32_2 val) // count 1
```

这些函数返回在要计数的“val”中的连续的 0 的个数（位数）的值。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

“_bseq0(x)”：

这是检测从输入的 MSB 开始有多少个连续的位 0 的函数。其格式如下：

```
int_bseq0(FIX16_1 val) // count 0
```

```
int_bseq0(FIX16_2 val) // count 0
```

```
int_bseq0(FIX32_1 val) // count 0
```

```
int_bseq0(FIX32_2 val) // count 0
```

这些函数返回在要计数的“val”中的连续的 0 的个数（位数）

的值。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

“_bseq1(x)”:

这是检测从输入的 MSB 开始有多少个连续的位 1 的函数。其格式如下:

```
int_bseq1(FIX16_1 val) // count 1
int_bseq1(FIX16_2 val) // count 1
int_bseq1(FIX32_1 val) // count 1
int_bseq1(FIX32_2 val) // count 1
```

这些函数返回在要计数的 “val” 中的连续的 1 的个数 (位数) 的值。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

“_bseq(x)”:

这是检测从输入的 MSB 的下一位开始有多少个连续的与 MSB 具有相同值的位的函数。其格式如下:

```
int_bseq(FIX16_1 val)
int_bseq(FIX16_2 val)
int_bseq(FIX32_1 val)
int_bseq(FIX32_2 val)
```

这些函数返回 “val” 中的标准化的位的个数。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

“_bcnt1(x)”:

这是检测在输入的所有位中包括多少位 1 的函数。其格式如下：

```
int_bcnt1(FIX16_1 val)
```

```
int_bcnt1(FIX16_2 val)
```

```
int_bcnt1(FIX32_1 val)
```

```
int_bcnt1(FIX32_2 val)
```

这些函数返回在要计数的“val”中的 1 的个数的值。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

“_extr(a,i1,i2)”：

这是提取输入的预定位位置并进行符号扩展的函数。其格式如下：

```
int_extr(FIX16_1 val1, int val2, int val3)
```

```
int_extr(FIX16_2 val1, int val2, int val3)
```

```
int_extr(FIX32_1 val1, int val2, int val3)
```

```
int_extr(FIX32_2 val1, int val2, int val3)
```

这些函数返回从位位置 val2 到位位置 val3 所指示的 val1 的位字段被提取和符号扩展的结果。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

“_extru(a,i1,i2)”：

这是提取输入的预定位位置并进行零扩展的函数。其格式如下：

```
unsigned int_extru(FIX16_1 val, int val2, int val3)
```

```
unsigned int_extru(FIX16_2 val, int val2, int val3)
```

```
unsigned int_extru(FIX32_1 val, int val2, int val3)
```

```
unsigned int_extru(FIX32_2 val, int val2, int val3)
```

这些函数返回从位位置 `val2` 到位位置 `val3` 所指示的 `val1` 的位字段被提取和零扩展的结果。对应于这些函数的机器语言指令被定义在内置函数定义文件 103 中。

(2) 被变换成两个或更多机器语言指令（一个机器语言指令序列）的内置函数

“`_modulo_add()`”:

这是执行模数寻址的地址更新的函数。其格式如下:

```
_modulo_add(void *addr, int imm, int mask, size_t size, void *base)
```

这里，每个变元的含义如下:

`addr`: 更新之前的地址或低位地址（模数部分）

`imm`: 相加值（数据的个数）

`mask`: 屏蔽的宽度（模数的宽度）

`size`: 数据的大小（2 的幂）

`base`: 基地址（阵列的首部地址）

这个函数返回由模数寻址只从地址 `addr` 加上相加值 `imm` 的结果。

对应于这个函数的机器语言指令被定义在内置函数定义文件 103 中。换句话说，这个函数使用用第二输入的预定位字段替代第一输入的预定位字段的指令（`addmsk`）来计算模数寻址。一个用法例子如下:

```
int array[MODULO];
```



```
p = array;

for (i = 0; i < 100; i++) {

*q++ = *p;

p = (int *)_modulo_add(p, 1, N, sizeof(int), array);

}
```

这里，变量 `MODULO` 是 2 的幂 (2^N)。在这个用法例子中，阵列的 100 个单元由 `MODULO*SIZE` 字节的对准来定位。

“`_brev_add()`”:

这是执行位反转寻址的地址更新的函数。其格式如下:

```
_brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
```

这里，每个变元的含义如下:

`addr`: 更新之前的地址

`cnt`: 位反转计数器

`mm`: 相加值 (数据的个数)

`mask`: 屏蔽的宽度 (反转的宽度)

`size`: 数据的大小 (2 的幂)

`base`: 基地址 (阵列的首部地址)

这个函数返回由位反转寻址只从对应于位反转计数器 `cnt` 的地址 `addr` 加上相加值 `mm` 的结果。

对应于这个函数的机器语言指令被定义在内置函数定义文件 103 中。换句话说，这个函数使用执行到第一输入的预定位字段的逐位的位置反转的指令 (`mskbrvh`) 来计算位反转寻址。一个用法例子如

下：

```
int array[BREV];

p = array;

for(i = 0; i < 100; i++) {

*q++ = *p;

p = (int *)_brev_add(p, i, 1, N, sizeof(int), array);

}
```

这里，变量 BREV 是 2 的幂 (2^N)。在这个用法例子中，阵列的 100 个单元用 BREV*SIZE 字节的对准来定位。

(3) 可以指定不是寄存器分配的资源的资源（例如累加器）的函数

在内置函数定义文件 103 中，除了作为优化中的寄存器分配的目标资源的通用寄存器之外，还准备了 (i) 是更新不是寄存器分配的目标（是隐含资源）的累加器的运算（乘法和乘积和运算）的以及 (ii) 可以用一个累加器作为基准类型指定一个临时变量的内置函数（乘法：“_mul”和乘积和运算：“_mac”）。具体格式分别如下：

```
_mul(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
```

这个函数将变量 a 和变量 b 一起相乘，将作为结果的 64 位数据的高 32 位设置到用于乘法的高位累加器 MH，将 64 位数据的低 32 位设置到用于乘法 ML 的低位累加器 ML，并且，进一步，将组合累加器 MH 的低 16 位和累加器 ML 的高 16 位的 32 位数据设置到变量 c。

```
_mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
```

这个函数将用于乘法的高位累加器 MH 和用于乘法的低位累加器 ML 的 64 位数据与通过将变量 a 和变量 b 相乘而获得的结果相加，并将作为结果的 64 位数据的高 32 位设置到用于乘法的高位累加器 MH，将 64 位数据的低 32 位设置到用于乘法 ML 的低位累加器 ML，并且，进一步，将组合累加器 MH 的低 16 位和累加器 ML 的高 16 位的 32 位数据设置到变量 c。

一个用法例子如下：

机器语言指令替代单元 122 遵循在内置函数定义文件 103 中的定义将下面的源程序

```
long mh, ml;
_mul (mh, ml, dummy, a, b);
_mac (mh, ml, e, c, d);
```

用下面的机器语言指令

```
mul m0, Rx, Ra, Rb
mov r0, mh0
mov r1, mh1
mov mh0, r0
mov mh1, r1
mac m0, Re, Rc, Rd, m0
```

来替代。

应该注意，在上面提到的机器语言指令序列中，第一到第三行

对应于函数 `_mul`，第四到第六行对应于函数 `_mac`。通过在优化单元 130 中去除冗余来删除象这样的第二到第五行机器语言指令序列，机器语言指令被优化到下面的机器语言指令序列

```
mul m0, Rx, Ra, Rb
```

```
mac m0, Re, Rc, Rd, m0。
```

如上所述，当使用可以指定不是寄存器分配的目标的资源（例如累加器）的函数时，通过编译程序（优化单元 130）内的优化删除一组定义（值的存储）和使用（对该值的引用）是非常可能的，因此，象这样的内置函数在优化方面也是有效的。

接着，解释本编译程序 100 的特征行为中的优化单元 130 的行为。

图 74 是显示优化单元 130 的变元优化单元 131 的行为的流程图。为了依赖变元产生一个合适的指令或一个合适的序列（算法），变元优化单元 131 产生：

（1）当所有函数变元是常数时（步骤 S110 的左边），使用通过下面设置的常数中折叠（fold）而获得的常数值作为运算数的机器语言指令（步骤 S111）；

（2）当变元的一部分是常数时（步骤 S110 的中间），立即值运算数的机器语言指令（步骤 112）；以及

（3）当所有变元是变量时（步骤 S110 的右边），寄存器运算数的机器语言指令序列（步骤 113）。

例如，当所有变元是常数时，象

```
d=_extru(0xffff, 7, 4);
```

产生一个具有通过在常数中折叠而获得的常数值机器语言指令，象

```
mov Rd, oxf.
```

另一方面，当变元的一部分是常数时，象

```
d=_extru(a, 7, 4);
```

产生一个立即运算数的机器语言指令，象

```
extru Rd, Ra, 7, 4。
```

进一步，当所有变元是变量时，象

```
d=_extru(a, b, c);
```

产生一个机器语言指令序列，象

```
aslRe, Rb, 8
```

```
andRf, Rc, 0x1f
```

```
orRg, Re, Rf
```

```
extruRd, Ra, Rg。
```

如刚才所描述的，从一个内置函数，不总是固定地产生相同的机器语言指令，而是由变元优化单元 131 产生依赖于变元性质而优化的机器语言指令（或机器语言指令序列）。

图 75 是显示一个算术树以解释优化单元 130 的类型变换优化单元 132 的行为的示意图。类型变换优化单元 132 对源程序中的特定记号的运算产生一个带有类型变换的机器语言指令（诸如 `fmulhw`），以执行不同类型之间的有效运算。

在普通 C 语言中，16 位×16 位的结果的类型是 16 位。存在 16 位×16 位→32 位的指令，但如下所述产生两个不同的机器语言指令。例如，对于

`f32=f16*f16;` 的描述，

产生两个指令：

`fmulhh // 16bit × 16bit -> 16bit`

`asl // 16bit -> 32bit 的类型变换`

因此，当在源程序中描述(FIX32)16 位*(FIX32)16 位时，类型变换优化单元 132 平常产生如图 75A 所示的算术树（产生类型变换的线—cord），但通过将这个算术树变换成图 75B 中所示的算术树产生一个 16 位×16 位→32 位的指令（`fmulhw`）。

图 76 是显示一个采样程序的例子以解释等待时间优化单元 133 的行为的示意图。等待优化单元 133 基于一个关于在源程序 101 中建立的汇编程序指令（优化的 `asm` 语句）中的等待时间（周期数的指定）的命令执行对机器语言指令的时间调度（`schedule`），以便在特定部分中的行为或特定行为中只花费指定周期数的执行时间。

用户可以用两种类型的指定方法设置等待时间。

一种方法是指定附连在特定指令上的标号之间的等待时间，就象图 76A 所示的程序中的指定（`LATENCY L1, L2, 2;`）。在图 76A 的例子中，等待时间优化单元 133 执行对机器语言指令序列的分配的时间调度，以使得从处理器 1 执行指令 `wte` 开始直到执行指令 `rde` 只经过 2 个周期。

另一种方法是指定在访问扩展寄存器单元 80 的指令 (rd, wt, rde,wte) 上的直到指令下一次访问扩展寄存器单元 80 的等待时间,就象图 76B 所示的程序中的指定 (指令 wte 内的 LATENCY(2))。在图 76B 的例子中,等待时间优化单元 133 执行对机器语言指令序列的分配的时间调度,以便从处理器 1 执行指令 wte 和访问扩展存储器单元 80 开始直到再次访问扩展存储器单元 80 只经过 2 个周期。

采用象这样的等待时间的配置,可以在已经被成行 (in-line) 扩展的代码和还未被成行扩展的代码之间执行优化 (合并指令, 去除冗余, 分类指令和分配寄存器), 并保证所指定的指令或访问的等待时间。换句话说, 常规上, 用户必须明确地插入一个 nop 指令, 而当用户使用编译程序 100 时, 他所必须做的只是为一个必要的指令或一个必要的访问指定一个必要的等待时间。

图 77 是解释分析程序单元 110 的定点模式切换单元 111 的行为的示意图。

当定点模式切换单元 111 在源程序 101 中检测到一个编译指示来保存和返回定点模式 (例如, “#pragma_save_fxpmode func”) 时, 定点模式切换单元 111 产生一个机器语言指令来保存和返回处理器 1 的 PSR 31 的位 FXP。

应该注意, 作为关于作为前提的定点的说明, 存在_1 系统 (FIX16_1, FIX32_1) 类型和_2 系统 (FIX16_2, FIX32_2) 类型; 模式通过硬件 (处理器 1) 中的 PSR 31 的一位 (FXP) 切换; 以及, 进一步, 存在在一个函数中只能使用一单个系统的条件。

因此，作为用于切换和使用程序上的这两个系统的方法，给出一个规则来指定一个编译指示（“#pragma _save_fxpmode”函数名）作为可以由其他系统调用的函数。从而，定点模式切换开关单元 111 将对应于 FIX 类型的模式的保存和返回的代码插入到函数的头部和尾部。另外，搜索每个函数的 FIX 类型声明；决定通过哪一 FIX 类型声明来编译函数；以及，插入设置模式的代码。

图 77A 显示了一个带有编译指示的函数的例子。在图 77A 的右边写的注释是定点模式切换单元 111 的插入处理，其具体处理如图 77B 所示。

象这样的编译指示的一个应用例子如图 77C 所示。例如，关于四个函数 f11、f21、f22 和 f23，当函数 f11:_1 系统调用函数 f21:_2 系统；函数 f21:_2 系统调用函数 f22:_2 系统；函数 f22:_2 系统调用函数 f23:_2 系统时，由于可以由其他模式调用的唯一的函数是 f21，所以可以通过只对这个函数执行编译指示指定来切换到一个正常模式。

如上所述，采用依据本实施例的编译程序 100，通过在运算符定义文件 102、内置函数定义文件 103 和机器语言指令替代单元 122 之间的协作处理，用户可以声明和使用定点类型的模式 0 和模式 1 作为普通类型并有效产生处理器 1 通过在高级语言级调用内置函数执行的高功能机器语言指令。

另外，利用变元优化单元 131 对内置函数的变元的优化，产生带有有效运算数的机器语言指令。此外，利用类型变换优化单元 132

对类型变换的优化，将一个带有类型变换的运算变换成处理器 1 执行的一个高性能机器语言指令。此外，利用等待时间优化单元 133 对机器语言指令的时间调度，用户可以在不插入 nop 指令的情况下保证在特定指令之间的或到扩展寄存器的访问中的等待时间。

至此，已经根据实施例解释了依据本发明的编译程序，但本发明并不限于这个实施例。

例如，在本实施例中，定点的类型是 16 位或 32 位，小数点被放在 MSB 或其低数位，但本发明并不限于这样的格式，其定点是 8 位或 64 位并且其小数点被放在另一个数位的类型作为目标也是可以接受的。

另外，提供一个使用类库作为用于用户的开发支持工具的行为验证技术也是可以接受的。换句话说，如图 78A 所示，通常，依据本实施例对于目标机器（处理器 1）采用交叉编译程序（编译程序 100）来编译测试源和定义文件 102 和 103；通过为带有专用模拟器的处理器 1 执行所获得的机器语言程序执行行为验证。作为替换，如图 78B 所示，准备一个其目标是用于开发的主机（例如，由英特尔公司制造的处理器）的类库（将运算符定义文件 102 和内置函数定义文件 103 分别与主机而非处理器 1 的机器语言指令联系起来的定义文件）并与测试源、定义文件 102 和 103 一起由本机编译程序（诸如 Visual C++(R)）编译并且主机执行所获得的机器语言程序 as-is 是可接受的。从而，可以在类似环境中高速执行一个模拟并执行行为验证。

此外，在本实施例中，与专用于目标处理器的机器语言指令相联系的运算符和内置函数被作为主文件（定义文件 102 和 103）提供，但依据本发明的编译程序可以被配置为在编译程序本身中结合象这样的定义文件的信息。换句话说，依据本发明的编译程序是一个将上述定义文件 102 和 103 结合在其中的整体类型（integral-type）的程序，将编译程序配置为使得编译程序将源程序翻译成机器语言程序，该程序包括其中定义了对应于专用于一个目标处理器的机器语言指令的运算的运算定义信息，编译程序包括：分析程序步骤，分析源程序；中间编码变换步骤，将分析的源程序变换成中间编码；优化步骤，优化变换的中间编码；以及，代码产生步骤，将优化的中间编码变换成机器语言指令，其中，中间编码变换步骤包括：检测子步骤，检测中间编码中的一个是否引用在运算定义信息中定义的运算；以及，替代子步骤，当检测到中间编码时，用一个对应的机器语言指令替代中间编码，以及，在优化步骤中，优化中间编码，中间编码包括在替代子步骤中替代中间编码的机器语言指令。从而，用户不需要在源程序中包括定义文件。

图1

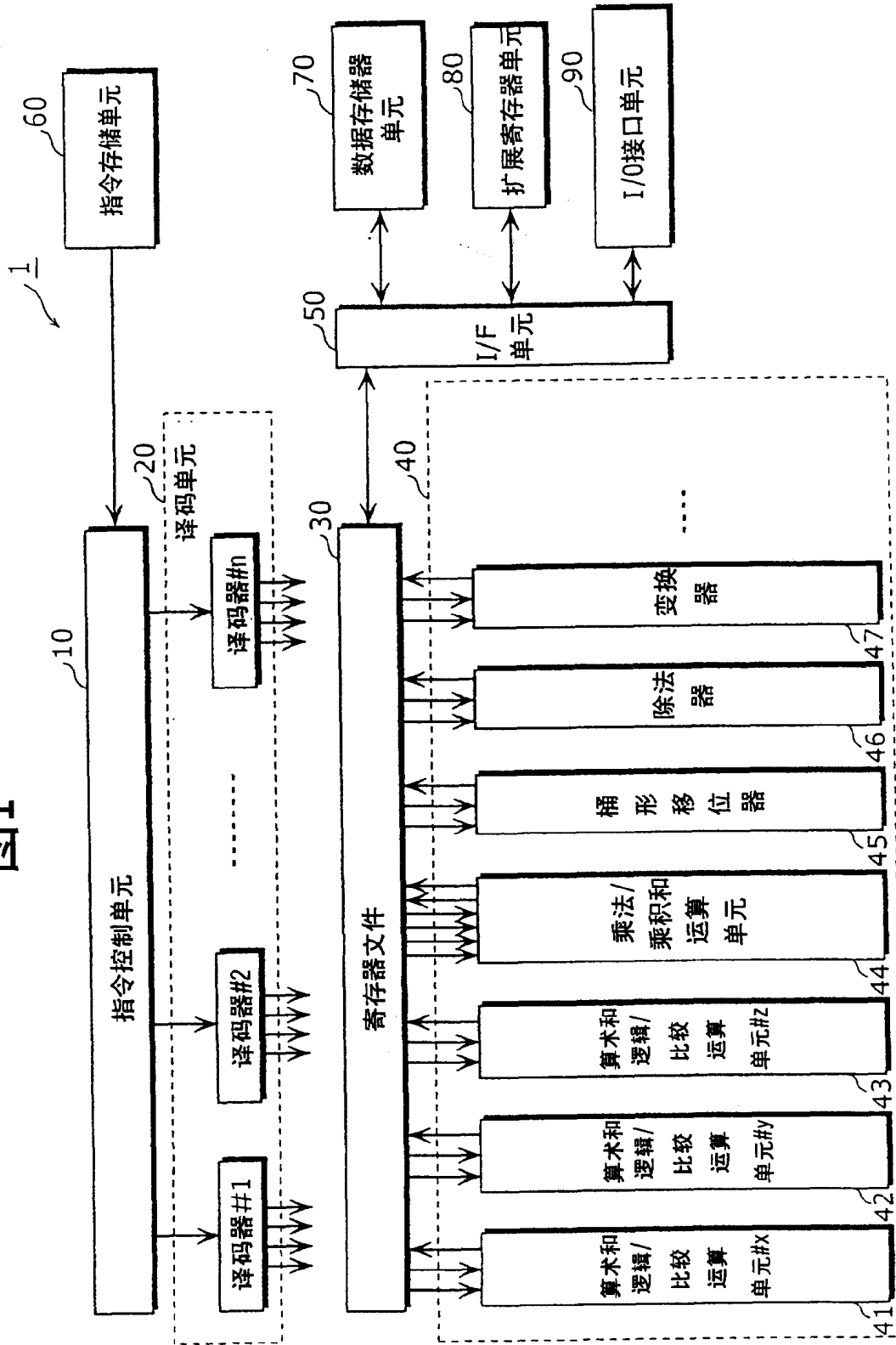


图2

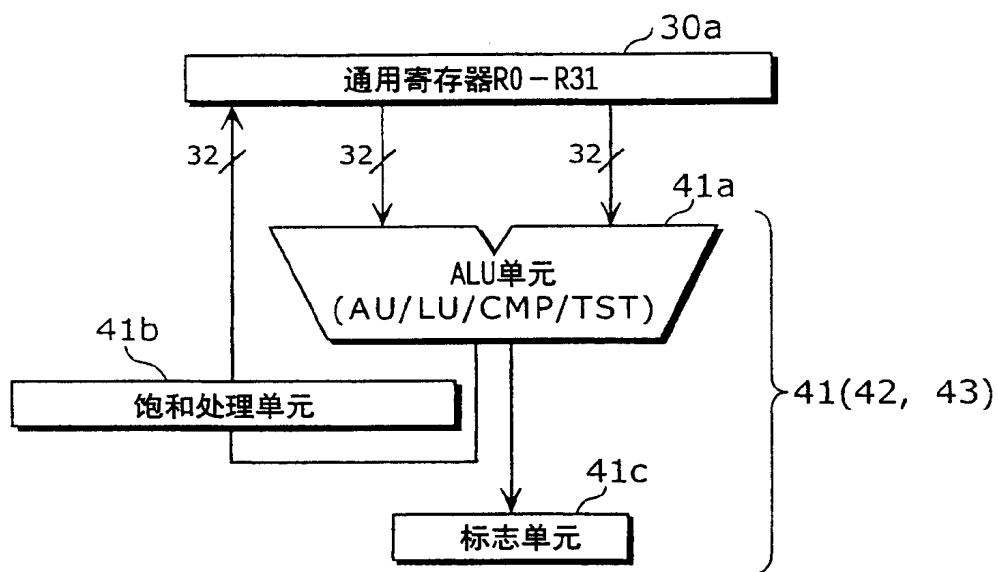


图3

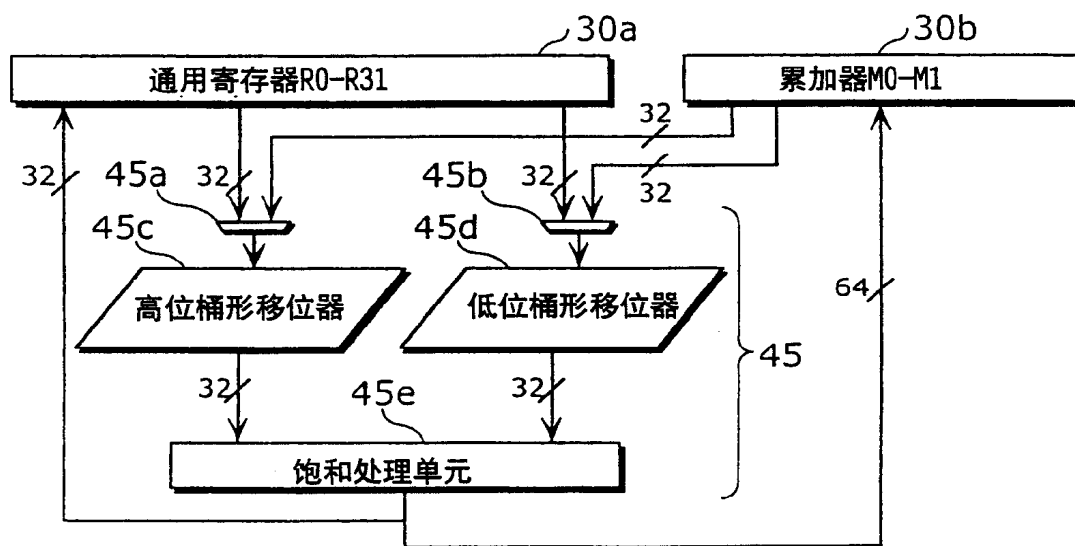


图4

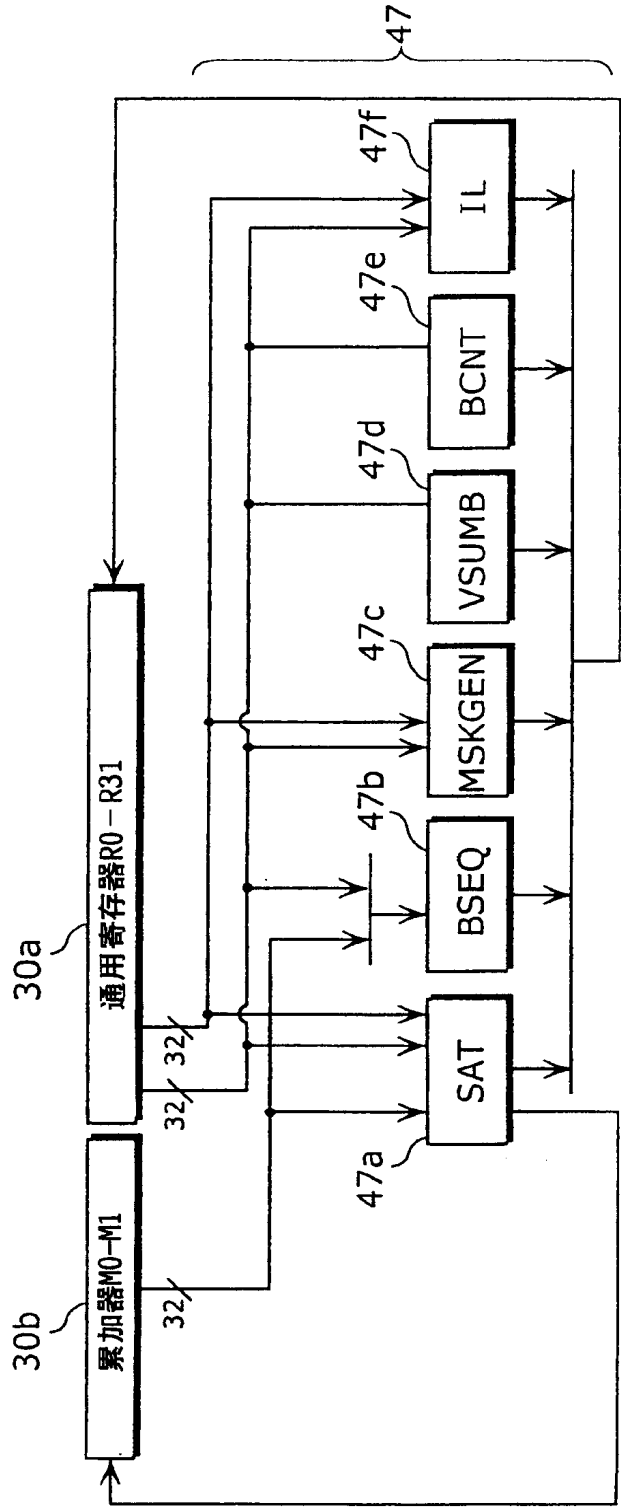


图5

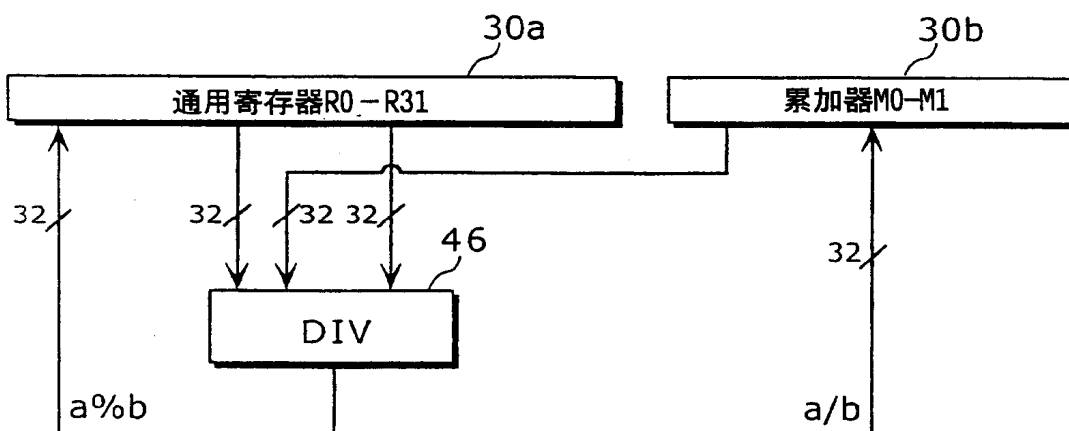


图6

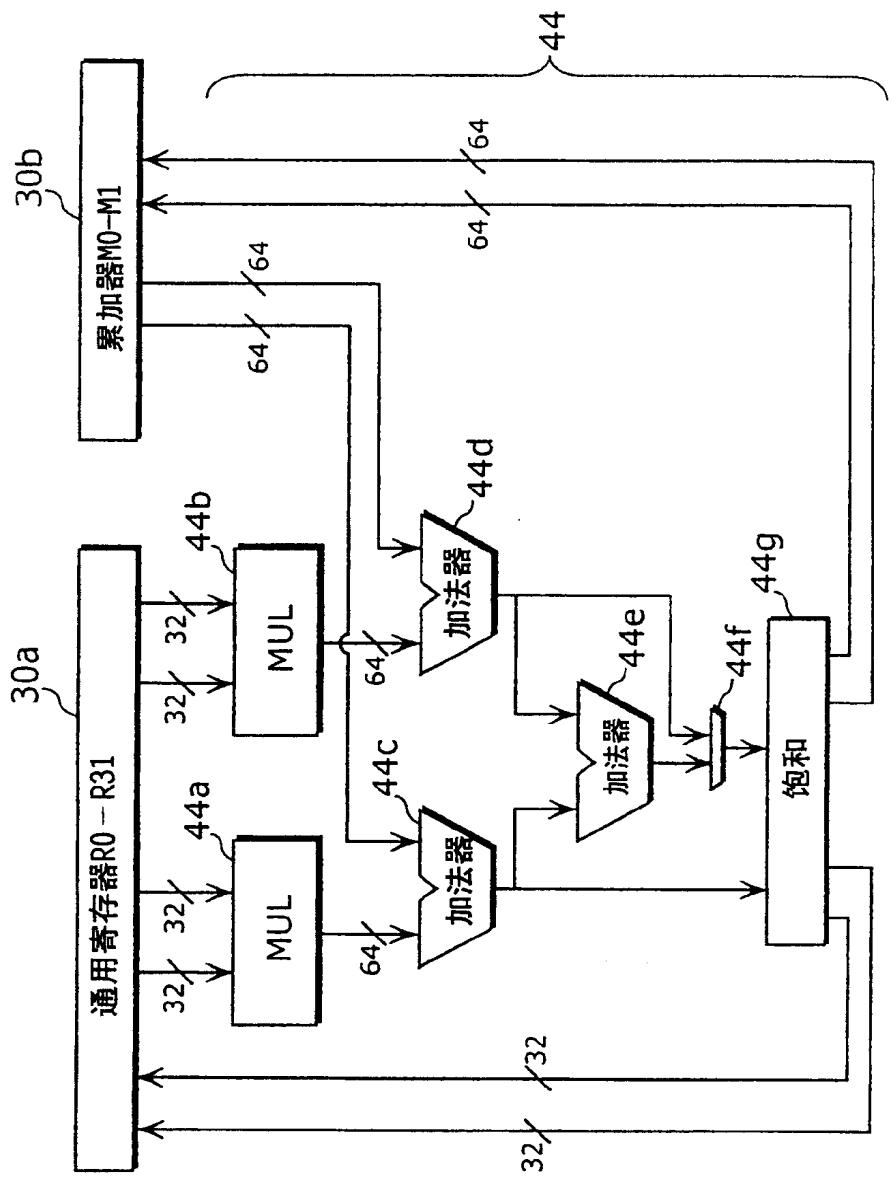


图7

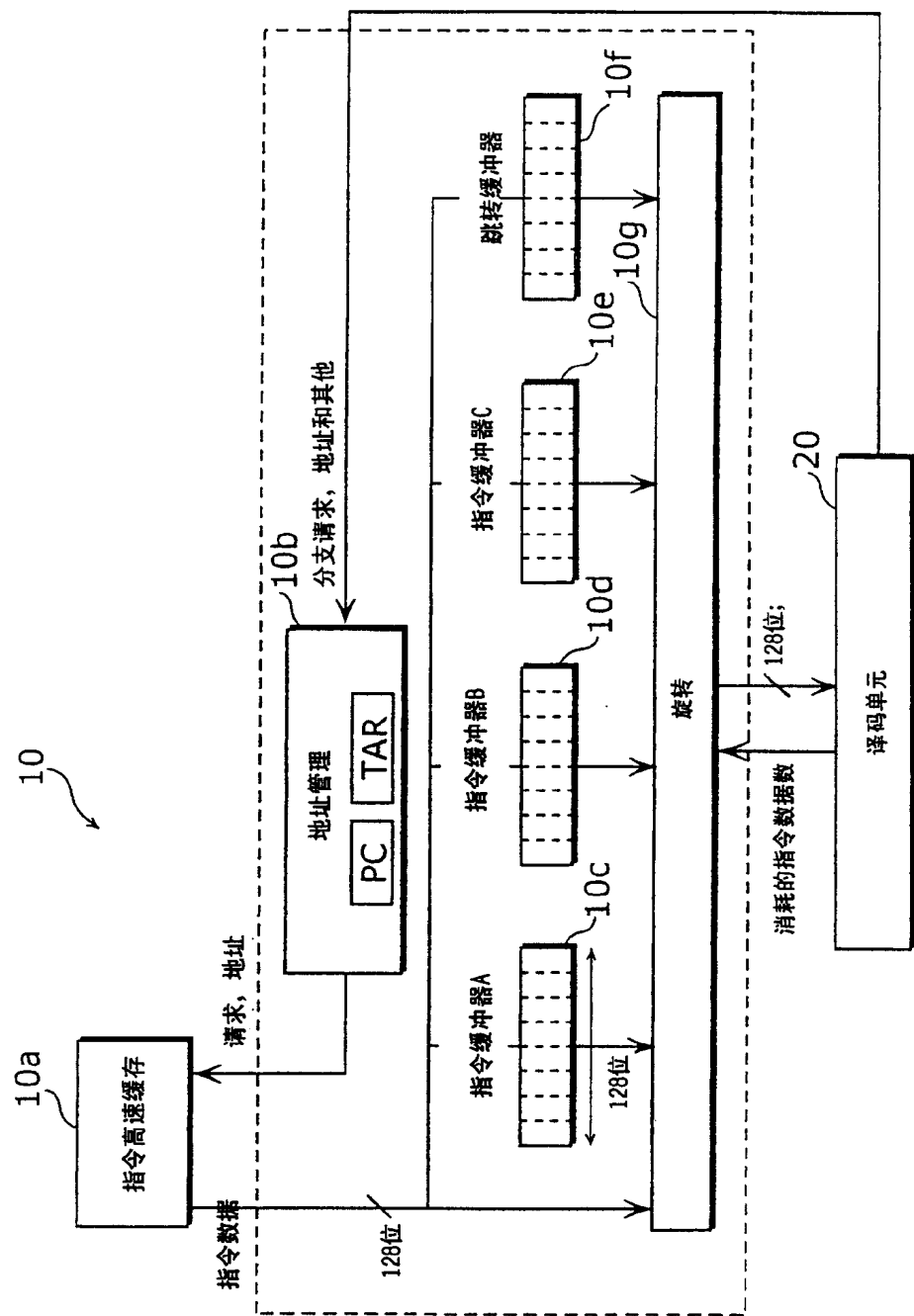


图8

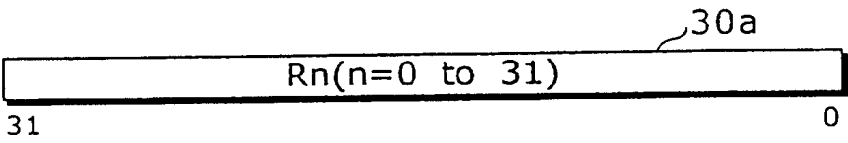


图9

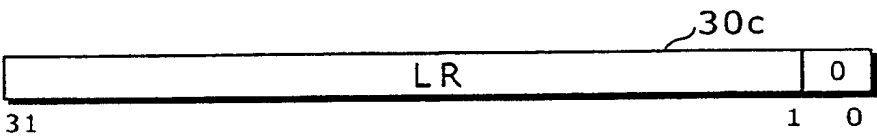


图10



图11

位	31	30	29	28	27	26	25	24
位名	保留	SWE	FXP	保留	IH	EH	PL	
位	23	22	21	20	19	18	17	16
位名	FIE3	FIE2	FIE1	FIE0	保留	保留	AEE	IE
位	15	14	13	12	11	10	9	8
位名	保留							
位	7	6	5	4	3	2	1	0
位名	IM[7:0]							

图12

32

位	31	30	29	28	27	26	25	24
位名	ALN		保留	BPO				
位	23	22	21	20	19	18	17	16
位名	保留			VC3		VC2	VC1	VC0
位	15	14	13	12	11	10	9	8
位名	保留						OVS	CAS
位	7	6	5	4	3	2	1	0
位名	C7	C6	C5	C4	C3	C2	C1	C0

图 13A



图 13B



30b

图 14

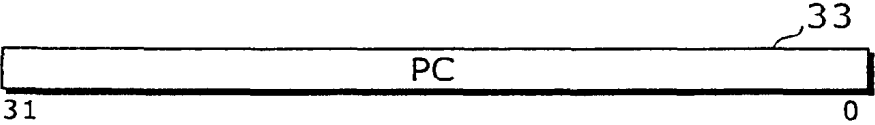


图 15

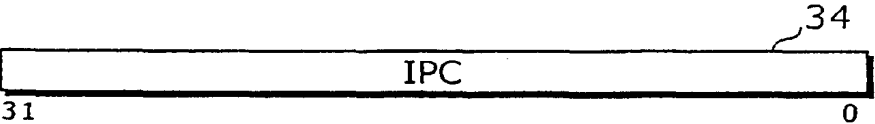


图 16



图17

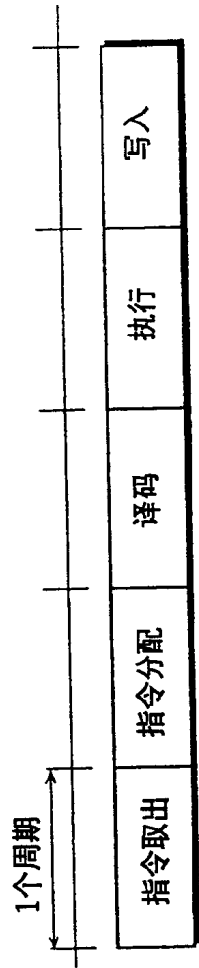


图18

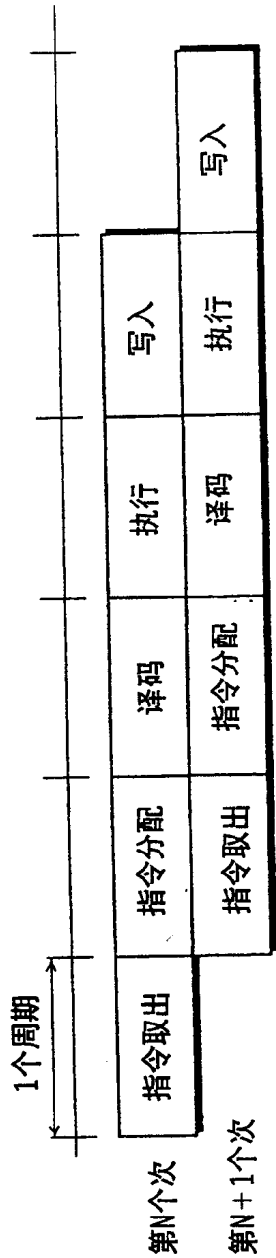


图19

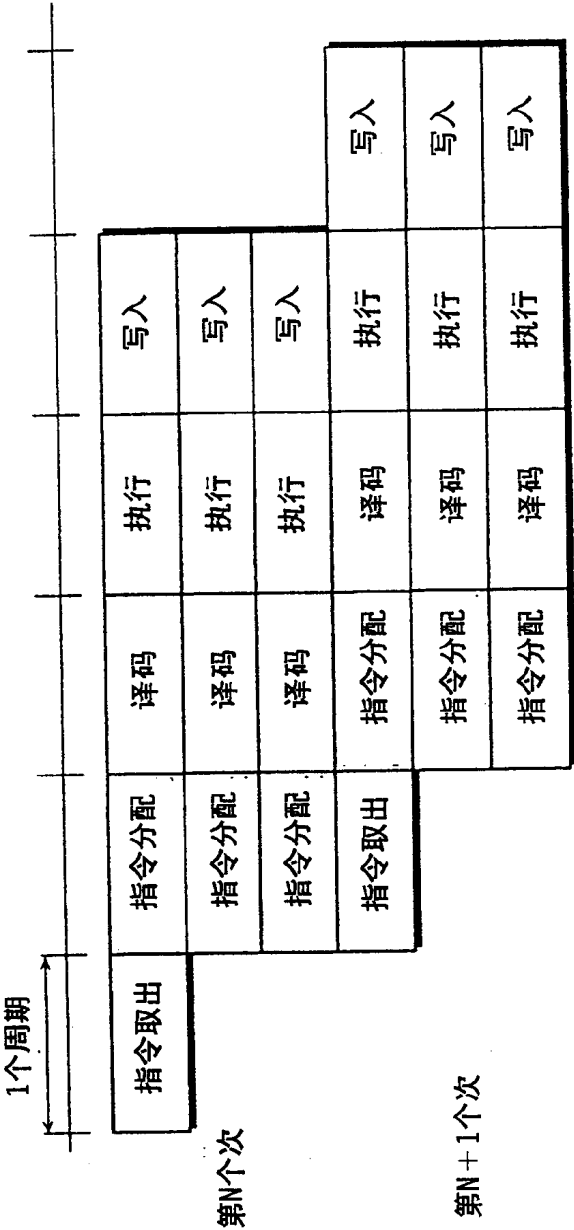


图20

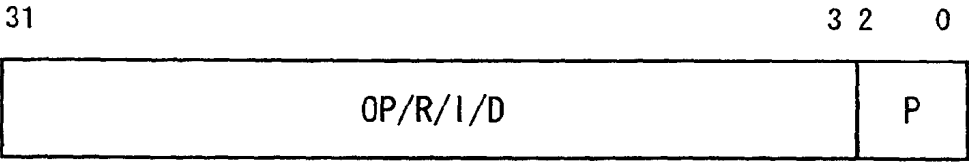


图21

类别	SDMO	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
ALU 加法 系统	S I N G L E	字	add	Rc,Ra,Rb Ra,Ra,12s SP,11s Ra2,Rb2 Rc3,Ra3,Rb3 Ra2,06s SP,11s					32 16
			addu	Ra,CP,18u Ra,SP,18u Ra3,SP,08u					32 16
			addc	Rc,Ra,Rb	Wcra,c0c1		带进位的加法		
			addvw	Rc,Ra,Rb	Wcra		带溢出的加法		
			adds	Ra,Ra,Rb			$Ra + Ra \rightarrow Ra$		32
			addar	Rc,Ra,Rb			$Ra + Ra + 1 \rightarrow Ra$		
			s1add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + Ra \rightarrow Ra$		16
			s2add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + Ra \rightarrow Ra$		32
			addmsh	Rc,Ra,Rb	RBP0		$Ra \rightarrow Ra$		16
			addervw	Rc,Ra,Rb					
		半个字	fsddvh	Rc,Ra,Rb	Wcra				
			vdh	Rc,Ra,Rb	Wcra		$Ra + Ra \rightarrow Ra$		
			vdhvh	Rc,Ra,Rb	Wcra		$Ra + Ra \rightarrow Ra$		
			vsddh	Ra,Ra,08s			$Ra + Ra \rightarrow Ra$		
			vdesh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddrh	Ra,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vdhvhc	Rc,Ra,Rb	RVC		$Ra + Ra \rightarrow Ra$		
			vdhvhc	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddhvh	Rc,Ra,Rb	Wcra		$Ra + Ra \rightarrow Ra$		
	S I M D	半个字	vhddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vhddhvh	Rc,Ra,Rb	Wcra		$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddhvh	Rc,Ra,Rb	Wcra		$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddhvh	Rc,Ra,Rb	Wcra		$Ra + Ra \rightarrow Ra$		
		字节	vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddh	Ra,Ra,08s			$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		
			vsddh	Rc,Ra,Rb			$Ra + Ra \rightarrow Ra$		

图22

类别	SMAD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 18 16 32 16
ALU 加法系统	SINGL	字	sub	Rc,Rb,Ra Rb2,Ra2 Rc3,Rb3,Ra3					
			rsub	Rb,Ra,08s Ra2,Rb2 Ra2,04s			$Rb - Ra \rightarrow Rb$ (Rb)		
			subc	Rc,Rb,Ra	Wcsrc0rc1		带进位		
			subwv	Rc,Rb,Ra	Wcsrc		带溢出		
			subwv	Rc,Rb,Ra					
		半个字	submah	Rc,Rb,Ra	R.BPD		$Rb - Ra \rightarrow Rb$ Rb		
			fsubvth	Rc,Rb,Ra					
			vsubh	Rc,Rb,Ra			$Rb - Ra \rightarrow Rb$ Rb		
			vsubhth	Rc,Rb,Ra	Wcsrc				
			vsubh	Rb,Ra,08s			立即值-立即值- $Rb - Ra \rightarrow Rb$ Rb		
	SIMD	半个字	vsubh	Rc,Rb,Ra			$Rb - Ra \rightarrow Rb$ Rb		
			vsubhth	Rc,Rb,Ra	Wcsrc				
			vsubh	Rc,Rb,Ra			$Rb - Ra \rightarrow Rb$ Rb		
			vsubhth	Rc,Rb,Ra	Wcsrc				
			vsubh	Rc,Rb,Ra			$Rb - Ra \rightarrow Rb$ Rb		
			vsubhth	Rc,Rb,Ra	Wcsrc				
			vsubh	Rc,Rb,Ra			$Rb - Ra \rightarrow Rb$ Rb		
			vsubhth	Rc,Rb,Ra	Wcsrc				
			vsubh	Rc,Rb,Ra			$Rb - Ra \rightarrow Rb$ Rb		
			vsubhth	Rc,Rb,Ra	Wcsrc				
		字节	vsubb	Rc,Rb,Ra			(立即值) $Rb - Ra \rightarrow Rb$ Rb		
			vsubb	Rb,Ra,08s			(带四舍五入) $Rb - Ra \rightarrow Rb$ Rb		
			vsubb	Rc,Rb,Ra	RVC				

图23

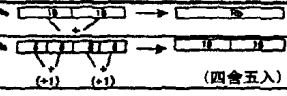
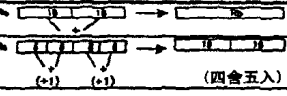
类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
ALU logic 系统	SINGLE	字	and	Rc, Ra, Rb $Rb, Ra, 0bu$ $Ra2, Rb2$			与 (AND)	A	32
			andn	Rc, Ra, Rb $Rb, Ra, 0bu$ $Ra2, Rb2$			或 (OR)		16 32
			or	Rc, Ra, Rb $Rb, Ra, 0bu$ $Ra2, Rb2$					16 32
			xor	Rc, Ra, Rb $Rb, Ra, 0bu$ $Ra2, Rb2$			异或		16 32
ALU MOV 系统	SINGLE	字	mov	$Rb, Reg32$ $Reg32, Rb$ $Rb2, Reg16$ $Reg16, Rb2$ $Ra2, Rb$ $Ra2, 0bu$			$Reg32 = TAR\ LR\ SVR\ PSR\ CFR\ MH0\ MH1\ MLD$ $ML1\ EPSR\ IPC\ IPSR\ PC\ EPC\ PSR0$ $PSR1\ PSR2\ PSR3\ CFR0\ CFR1\ CFR2\ CFR3$ $Reg16 = TAR\ LR\ MH0\ MH1$	A	32
			movp	$Rc, Rb+1, Ra, Rb$			$Rc \leftarrow Rb, Rb+1 \leftarrow Rb$		16
			movcf	Cc, Cc, Cc, Cc			$Cc \leftarrow Cc, Cc \leftarrow Cc$		32
			movcvs	$Cmc, Cmc+1$	Wave		$Cmc, Cmc+1 \leftarrow Cmc, Cmc+1$		16
			movccas	$Cmc, Cmc+1$	Wave		$Cmc, Cmc+1 \leftarrow Cmc, Cmc+1$		32
			satN	$Ra, 16s$					32
ALU MAX MIN 系统	SINGLE	字	max	Rb, Ra, Rb	Wave		$Rc \leftarrow \max(Ra, Rb)$	A	32
		半个字	min	Rb, Ra, Rb	Wave		$Rc \leftarrow \min(Ra, Rb)$		32
		字节	vmash	Rb, Ra, Rb					32
			vmash	Rb, Ra, Rb					32
ALU ABS 系统	SINGLE	字	abs	Rb, Rb			绝对值	A	32
		半个字	absrv	Rb, Rb	Wave		带符号输出		32
		字节	absrvh	Rb, Rb	Wave				32
			absrvh	Rb, Rb	Wave				32
ALU NEG 系统	SINGLE	字	negrv	Rb, Rb	Wave			A	32
		半个字	negrvh	Rb, Rb	Wave				32
		字节	negrvh	Rb, Rb	Wave				32
			negrvh	Rb, Rb	Wave				32
ALU SUM 系统	SINGLE	半个字	vsunh	Rb, Rb				A	32
		半个字	vsunh2	Rb, Rb					32
		字节	vsunh2	Rb, Rb			(四舍五入)		32
			vsunh2	Rb, Rb					32
ALU RND 系统	SINGLE	半个字	rndvh	Rb, Rb	Wave		四舍五入	C	32
		半个字	rndvh	Rb, Rb	Wave				32
		字节	rndvh	Rb, Rb	Wave				32
			rndvh	Rb, Rb	Wave				32

图24

类别	SMC	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
CMP	S I N G L E		cmpCCn	Cm,Ra,Rb,Cn Cm,Ra,i05s,Cn Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05s,Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, le, lt, leu, leu Cm <- result & Cn (Cm+1 <- ~result & Cn)	A	32
			cmpCCa	Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05s,Cn	W:CF		Cm <- result & Cn Cm+1 <- ~result & Cn		
			cmpCCo	Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05s,Cn	W:CF		Cm <- result Cn Cm+1 <- ~result Cn		
			cmpCC	C6,Ra2,Rb2 C6,Ra2,i04s	W:CF		CC = eq, ne, gt, ge, le, lt C6 <- result		16
			testzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn (Cm+1 <- ~(Ra & Rb == 0) & Cn)		32
			testza	Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn Cm+1 <- ~(Ra & Rb == 0) & Cn		
			testzo	Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) Cn Cm+1 <- ~(Ra & Rb == 0) Cn		
			testvn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn (Cm+1 <- ~(Ra & Rb != 0) & Cn)		
			testna	Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn Cm+1 <- ~(Ra & Rb != 0) & Cn		
			testno	Cm,Cm+1,Ra,Rb,Cn Cm,Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) Cn Cm+1 <- ~(Ra & Rb != 0) Cn		
			testz	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		CC <- (Ra2&Rb2 == 0)		
			testn	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		CC <- (Ra2&Rb2 != 0)		16
S I M D		半个字	vcmpCCh	Ra,Rb	W:CF		CC = eq, ne, gt, le, ge, lt		32
			vscompCCh	Ra,i05s					
		字节	vcmpCCb	Ra,Rb	W:CF		CC = eq, ne, gt, le, ge, lt		
			vscompCCb	Ra,i05s					

图25

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
mul 系统	S I N G L E	字 × 字	mul	Min,Rc,Ra,Rb Min,Rb,Ra,08s				X2	
			mulu	Min,Rc,Ra,Rb Min,Rb,Ra,08s			无符号的乘法		
			fmulwv	Min,Rc,Ra,Rb		fup	定点运算		
		字 × 半个字	hmul	Min,Rc,Ra,Rb					
			lmul	Min,Rc,Ra,Rb					
			fmulhw	Min,Rc,Ra,Rb		fup			
		半个字 × 半个字	fmulhw	Min,Rc,Ra,Rb		fup		X1	
			fmulhh	Min,Rc,Ra,Rb		fup			
			fmulhvr	Min,Rc,Ra,Rb		fup	带四舍五入		
			vmul	Min,Rc,Ra,Rb					
			vfmulw	Min,Rc,Ra,Rb		fup			
			vfmulh	Min,Rc,Ra,Rb		fup			
	S I M D	半个字 × 半个字	vfmulhr	Min,Rc,Ra,Rb		fup	带四舍五入	X2	32
			vsmul	Min,Rc,Ra,Rb					
			vfmulw	Min,Rc,Ra,Rb		fup			
			vfmulh	Min,Rc,Ra,Rb		fup			
			vfmulhr	Min,Rc,Ra,Rb		fup	带四舍五入		
			vsmul	Min,Rc,Ra,Rb					
			vfmulw	Min,Rc,Ra,Rb		fup			
			vfmulh	Min,Rc,Ra,Rb		fup			
			vfmulhr	Min,Rc,Ra,Rb		fup	带四舍五入		
			vsmul	Min,Rc,Ra,Rb					
			vfmulw	Min,Rc,Ra,Rb		fup			
			vfmulh	Min,Rc,Ra,Rb		fup			
			vfmulhr	Min,Rc,Ra,Rb		fup	带四舍五入		
			vsmul	Min,Rc,Ra,Rb					
			vfmulw	Min,Rc,Ra,Rb		fup			
			vfmulh	Min,Rc,Ra,Rb		fup			
			vfmulhr	Min,Rc,Ra,Rb		fup	带四舍五入		
			vsmulwv	Min,Rc,Rc+1,Ra,Rb Min,Rc,Rc+1,Ra,Rb		fup			

图26

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
mac 系统	S I N G L E	字 × 字	mac	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用mul的乘积和运算	X2	32
			fmacww	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用fmulww的乘积和运算		
		字 × 半个字	hmac	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用hmul的乘积和运算	X1	16
			lmac	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用lmul的乘积和运算		32
		半个字 × 半个字	fmachww	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用fmulhww的乘积和运算		16
			fmachw	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用fmulhw的乘积和运算		32
			fmachh	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用fmulhh的乘积和运算		
			fmachhr	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	四舍五入		
	S I M D	半个字 × 半个字	vmac	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用vmul的乘积和运算	X2	16
			vfmacw	Mn, Rc, Ra, Rb, Mn		fxp	采用vfmulw的乘积和运算		32
			vsmac	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用vxmul的乘积和运算		16
			vxfmacw	Mn, Rc, Ra, Rb, Mn		fxp	采用vxfmulw的乘积和运算		32
			vxfmach	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用vxfmulh的乘积和运算		
			vxfmachr	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	带四舍五入		
			vhmec	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用vxfmul的乘积和运算		
			vhfmacw	Mn, Rc, Ra, Rb, Mn		fxp	采用vhmulw的乘积和运算		
			vhfmac	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用vhfmulh的乘积和运算		
			vhfmacr	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	带四舍五入		
			vmec	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz			采用vmul的乘积和运算		
			vfmecw	Mn, Rc, Ra, Rb, Mn		fxp	采用vfmulw的乘积和运算		
			vfmec	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用vfmulh的乘积和运算		
			vfmecr	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	带四舍五入		
			vfmec	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	采用vlfmulh的乘积和运算		
			vlfmecr	Mn, Rc, Ra, Rb, Mn MQ, Rc, Ra, Rb, Rz		fxp	带四舍五入		
		字 × 半个字	vpfmachw	Mn, Rc, Rc+1, Ra, Rb, Mn		fxp			

图27

类别	SMD	尺寸	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
msu 系统	S I N G L E	字 × 字	msu	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			采用mul的乘积差运算	x2	32
			fmsuww	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	采用fmu1ww的乘积差运算		
		字 × 半个字	hmsu	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			采用hmul的乘积差运算	x1	
			imsu	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			采用lmu1的乘积差运算		
		半个字 × 半个字	fmsuhww	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	采用fmu1hww的乘积差运算		
			fmsuhw	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	采用fmu1hw的乘积差运算		
			fmsuhh	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	采用fmu1hh的乘积差运算		
			fmsuhv	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	带四舍五入		
		半个字 × 半个字	vmsu	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			采用vmul的乘积差运算	x2	
			vfmaww	Mn,Rc,Ra,Rb,Mn		fxp	采用vfmul的乘积差运算		
			vfmah	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	采用vfmulh的乘积差运算		
			vmsu	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			采用vxmul的乘积差运算		
vxfmaww	Mn,Rc,Ra,Rb,Mn			fxp	采用vxfmu1w的乘积差运算				
vxfmah	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			fxp	采用vxfmu1h的乘积差运算				
vhmaww	Mn,Rc,Ra,Rb,Mn			fxp	采用vhmu1w的乘积差运算				
vfhmah	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			fxp	采用vhfmulh的乘积差运算				
vmsu	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				采用v1mul的乘积差运算				
vfmaww	Mn,Rc,Ra,Rb,Mn			fxp	采用v1fmulw的乘积差运算				
vfmah	Mn,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			fxp	采用vfmulh的乘积差运算				

图28


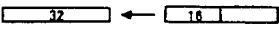
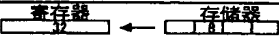
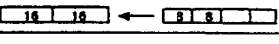
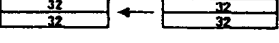
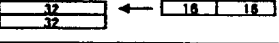
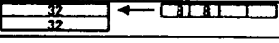

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16		
MEM 1d 系统	S I M D	字	ld	Rb,(Ra,d10u) Rb,(GP,d13u) Rb,(SP,d13u) Rb,(Ra+)*10s Rb2,(Ra2) Rb2,(Ra2,d05u) Rb2,(GP,d06u) Rb2,(SP,d06u) Rb2,(Ra2+)				M	32		
			半个字	ldh	Rb,(Ra,d09u) Rb,(GP,d12u) Rb,(SP,d12u) Rb,(Ra+)*09s Rb2,(Ra2) Rb3,(Ra3,d04u) Rb2,(GP,d05u) Rb2,(SP,d05u) Rb2,(Ra2+)					32	
				字节	ldb	Rb,(Ra,d08u) Rb,(GP,d11u) Rb,(SP,d11u) Rb,(Ra+)*08s					32
					ldbuh	Rb,(Ra,d08u) Rb,(GP,d11u) Rb,(SP,d11u) Rb,(Ra+)*08s					
		字节→ 半个字		ldbh ldbuh	Rb,(Ra+)*07s Rb,(Ra+)*07s					16	
			字	ldp	Rb:Rb+1,(Ra,d11u) LR:SVR,(Ra,d11u) TAR:UDR,(Ra,d11u) Rb:Rb+1,(GP,d14u) LR:SVR,(GP,d14u) TAR:UDR,(GP,d14u) Rb:Rb+1,(SP,d14u) LR:SVR,(SP,d14u) TAR:UDR,(SP,d14u) Rb:Rb+1,(Ra+)*11s Rb:Rb+1,(SP,d07u) LR:SVR,(SP,d07u) Rb2:Rb2,(Ra2+)					32	
				半个字	ldhsp	Rb:Rb+1,(Ra,d10u) Rb:Rb+1,(Ra+)*10s Rb2:Rb2,(Ra2+)					32
					字节	ldbsp	Rb:Rb+1,(Ra,d08u) Rb:Rb+1,(Ra+)*08s				
	字节→ 半个字	ldbhsp ldbuhsp				Rb:Rb+1,(Ra+)*07s Rb:Rb+1,(Ra+)*07s				32	
	P A I R										

图29

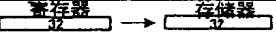
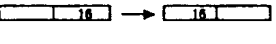

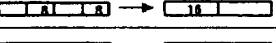

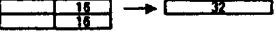
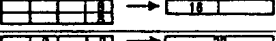

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
MEM store 系统	S I N G L E	字	st	(Ra,d10u),Rb (GP,d13u),Rb (SP,d13u),Rb (Ra+),Rb (Ra2),Rb2 (Ra2,d05u),Rb2 (GP,d06u),Rb2 (SP,d06u),Rb2 (Ra2+),Rb2					32
								16	
		半个字	sth	(Ra,d08u),Rb (GP,d12u),Rb (SP,d12u),Rb (Ra+),Rb (Ra2),Rb2 (Ra2,d04u),Rb2 (GP,d05u),Rb2 (SP,d05u),Rb2 (Ra2+),Rb2					32
								16	
	P A I R	字节	stb	(Ra,d08u),Rb (GP,d11u),Rb (SP,d11u),Rb (Ra+),Rb					
		字节→半个字	stbh	(Ra+),Rb					
		字	stp	(Ra,d11u),Rb:Rb+1 (Ra,d11u),LR:SVR (Ra,d11u),TAR:UDR (GP,d14u),Rb:Rb+1 (GP,d14u),LR:SVR (GP,d14u),TAR:UDR (SP,d14u),Rb:Rb+1 (SP,d14u),LR:SVR (SP,d14u),TAR:UDR (Ra+),Rb:Rb+1 (SP,d07u),Rb:Ra (SP,d07u),LR:SVR (Ra2+),Rb2:Ra2					32
								16	
			半个字	sthp	(Ra,d10u),Rb:Rb+1 (Ra+),Rb:Rb+1 (Ra2+),Rb2:Ra2				
	字节		stbp	(Ra,d08u),Rb:Rb+1 (Ra+),Rb:Rb+1					16
	字节→半个字	stbhp	(Ra+),Rb:Rb+1					32	

图30

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
BRA			setlr	d09s C5,d09s			设置LR 在分支缓冲器中存储从LR取出的指令	B	32
			settar	d09s C6,d09s C6,C2:C4,d09s C6,Cm,d09s C6,C4,d09s	W:c6 W:c2:c4,c6 W:c6,cm W:c6		设置TAR 在分支缓冲器中存储从TAR取出的指令		16
			setbb	LR TAR			在分支缓冲器中存储从LR取出的指令 在分支缓冲器中存储从TAR取出的指令		32
			jloop	C5,LR,Ra,i08s C6,TAR,Ra,i08s C6,C2:C4,TAR,Ra,i08s C6,Cm,TAR,Ra,i08s C6,TAR,Ra2 C6,C2:C4,TAR,Ra2 C6,Cm,TAR,Ra2	W:c5 W:c6 W:c2:c4,c6 W:c6,cm W:c6 W:c2:c4,c6 W:c6		仅仅判定[c5] 仅仅判定[c6]		16
			jmp	TAR LR					32
			jmp1	TAR LR	RCF				16
			jmpf	TAR LR Cm,TAR C6,C2:C4,TAR					32
			jmprr	LR					16
			br	d20s d08s			仅仅判定[c6] [c7]		32
			bri	d20s d09s	RCF				32
			rti			W-PSR Rsh			16

图31

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 18
BS asl 系统	S I N G L E	字	asl	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			左移 $\ll Rb[i0:4]$ 	S1	32 16
			faslvw	Rc,Ra,Rb Rb,Ra,i05u Rc,Ra,Rb Rb,Ra,i05u	Wzova		$\ll Rb[i0:4]$ 带饱和 		
		成对字	aslp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			$\ll Rb[i0:4]$ 	S2	
			faslpvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u	Wzova		$\ll Rb[i0:4]$ 带饱和 		
	S I M D	字节	vasl	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			$\ll Rb[i0:4]$ 	S2	32
			vfaslvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u	Wzova		$\ll Rb[i0:4]$ 带饱和 		
		半个字	vaslh	Rc,Ra,Rb Rb,Ra,i04u			$\ll Rb[i0:3]$ $\ll Rb[i0:3]$ 	S1	
			vfaslvh	Rc,Ra,Rb Rb,Ra,i04u	Wzova		$\ll Rb[i0:3]$ $\ll Rb[i0:3]$ 带饱和 		
		字节	vaslb	Rc,Ra,Rb Rb,Ra,i03u			$\ll Rb[i0:2]$ $\ll Rb[i0:2]$ 		
BS asr 系统	S I N G L E	字	asr	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			算术右移 $\gg Rb[i0:4]$ 	S1	32 16
		成对字	asrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			$\gg Rb[i0:4]$ 	S2	
	S I M D	字	vasr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			$\gg Rb[i0:4]$ 		32
		半个字	vasrh	Rc,Ra,Rb Rb,Ra,i04u			$\gg Rb[i0:3]$ $\gg Rb[i0:3]$ 	S1	
		字节	vasrb	Rc,Ra,Rb Rb,Ra,i03u			$\gg Rb[i0:2]$ $\gg Rb[i0:2]$ 		

图32

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16	
BS lsr 系统	S I N G L E	字	lsr	Rc,Ra,Rb Rb,Ra,i05u			逻辑右移 $\gg Ra[0:4]$ 	S1	32	
		成对字	lsrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,Mm,Ra,Rb Mm,Rb,Mm,Ra,i06u			$\gg Ra[0:4]$ 	S2		
	S I M D	字	vlslr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			$\gg Ra[0:4]$ $\gg Ra[0:4]$ $(1D) \gg Ra[0:4]$ $(2D) \gg Ra[0:4]$ 	S1		
		半个字	vlslh	Rc,Ra,Rb Rb,Ra,i04u			$\gg Ra[0:3]$ $\gg Ra[0:3]$ $Ra[0:3]$ $Ra[0:3]$ 	S1		
		字节	vlslb	Rc,Ra,Rb Rb,Ra,i03u			$\gg Ra[0:2]$ $\gg Ra[0:2]$ $Ra[0:2]$ $Ra[0:2]$ $Ra[0:2]$ $Ra[0:2]$ 	S1		
BS rotate 系统	S I N G L E	字	rol	Rc,Ra,Rb Rb,Ra,i05u			循环左移 $\ll Ra[0:4]$ 	S1	32	
	S I M D	半个字	vroih	Rc,Ra,Rb Rb,Ra,i04u				S1		
		字节	vroib	Rc,Ra,Rb Rb,Ra,i03u				S1		
BS ext 系统	S I N G L E	字	extw	Mm,Rb,Ra			$\gg Ra[0:4]$ $\gg Ra[0:4]$ 	C	32	
		半个字	exth	Ra2			$\gg Ra[0:3]$ $\gg Ra[0:3]$ $Ra[0:3]$ $Ra[0:3]$ 	S2		
			exthu	Ra2			$\gg Ra[0:2]$ $\gg Ra[0:2]$ $Ra[0:2]$ $Ra[0:2]$ 			
		字节	extb	Ra2			$\gg Ra[0:2]$ $\gg Ra[0:2]$ $Ra[0:2]$ $Ra[0:2]$ 	S2		
			extbu	Ra2			$\gg Ra[0:2]$ $\gg Ra[0:2]$ $Ra[0:2]$ $Ra[0:2]$ 			
	S I M D	半个字	vxexth	Mm,Rb,Ra			$\gg Ra[0:3]$ $\gg Ra[0:3]$ $Ra[0:3]$ $Ra[0:3]$ 	C	32	

图33

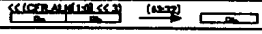
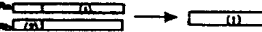
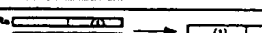
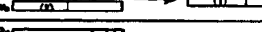
类别	SMID	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
CNV valn 系统	SIMD		valn	Rc,Ra,Rb	Rvaln[1:0]			C	32
			valn1	Rc,Ra,Rb					
			valn2	Rc,Ra,Rb					
			valn3	Rc,Ra,Rb					
			valnvc1	Rc,Ra,Rb	RVC0				
			valnvc2	Rc,Ra,Rb	RVC0				
			valnvc3	Rc,Ra,Rb	RVC0				
			valnvc4	Rc,Ra,Rb	RVC0				

图34

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
CNV	SINGLE		bcntl	Rb, Ra			对1的个数计数	C	32
			bseq0	Rb, Ra			从MSB开始对值的个数计数, 直到到达第一个0		
			bseq1	Rb, Ra			从MSB开始对值的个数计数, 直到到达第一个1		
			bseq	Rb, Ra			从MSB开始对值的个数计数, 直到到达第一个1		
			mskbrvh	Rc, Ra, Rb	RBP0				
			bytesrv	Rb, Ra					
			mskbrvb	Rc, Ra, Rb	RBP0				
	SIMD	半个字	vinthl	Rc, Ra, Rb					
			vinthh	Rc, Ra, Rb					
		字节	vinthb	Rc, Ra, Rb					
			vinthb	Rc, Ra, Rb					
		半个字	vlunpkh	Rb:Rb+1, Ra					
		字节	vlunpkb	Rb:Rb+1, Ra					
		半个字	vlunpkh	Rb:Rb+1, Ra					
			vlunpkh	Rb:Rb+1, Ra					
		字节	vlunpkb	Rb:Rb+1, Ra					
			vlunpkb	Rb:Rb+1, Ra					
		半个字	vnupk1	Rb, Min					
			vnupk2	Rb, Min					
			vstovh	Rb, Ra					
		字节	vstovb	Rb, Ra					
			vhpkb	Rc, Ra, Rb					

图35

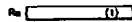
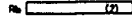
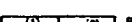

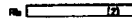
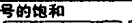
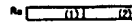

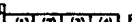


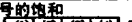
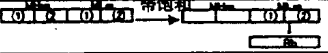
类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31 16
SAT vlpk 系统	SIMD	字-半个字	vlpkh	Rc,Ra,Rb			Ra  (0) 带饱和 Rb  (2) →  (0) (2) Rc	C	32
			vlpkhw	Rc,Ra,Rb			Ra  (0) 带无符号的饱和 Rb  (2) →  (0) (2) Rc		
		半个字-字节	vlpkb	Rc,Ra,Rb			Ra  (0) (2) 带饱和 Rb  (4) (6) →  (0) (2) (4) (6) Rc		
			vlpkbu	Rc,Ra,Rb			Ra  (0) (2) 带无符号的饱和 Rb  (4) (6) →  (0) (2) (4) (6) Rc		
SAT sat 系统	SINGLE	字	sacw	Mm,Rb,Mn			字饱和	C	32
			satb	Rb,Ra			半个字饱和		
			satb	Rb,Ra			字节饱和		
			satbu	Rb,Ra			无符号字节饱和		
			sat9	Rb,Ra			9位饱和		
			sat12	Rb,Ra			12位饱和		
	SIMD	半个字	vsath	Mm,Rb,Mn			 带饱和		
			vsath8	Rb,Ra			有符号的8位饱和		
			vsath8u	Rb,Ra			无符号的8位饱和		
			vsath9	Rb,Ra			9位饱和		
			vsath12	Rb,Ra			12位饱和		

图36

类别	SIMD	大小	指令	运算数	CFR	PSR	典型行为	运算单元	31/16
MSK			maskgen	Rc,Rb Rb,IOSU,IOSu			产生屏蔽 		16
			mask	Rc,Ra,Rb Rb,Ra,IOSU,IOSu				S2	32
EXTR			extr	Rc,Ra,Rb Rb,Ra,IOSU,IOSu			带符号扩展 	S2	32
			extru	Rc,Ra,Rb Rb,Ra,IOSU,IOSu			(不带符号扩展)		
DIV			div divu	MHm,Rc,MHn,Ra,Rb MHm,Rc,MHn,Ra,Rb	W.ovs		除法	DIV	32
ETC			pinM			W3hie,Re,pl R:PSR	软件中断 N=0~7	B	32
			pin			W3hie,pl R:PSR	软件中断 N=0~7		16
			scN			W3hie,pl R:PSR	系统调用		
			ldstb	Rb,(Ra)			屏蔽总线锁定	M	32
			rd	Rb,(Ra) Rb,(d11u) Rb2,(Ra2)		R:see	外部寄存器读出		16
			wt	(Ra),Rb (d11u),Rb (Ra2),Rb2		R:see	外部寄存器写入		32
			dpraf	(Ra,d11u)			预取		16
			dbgmn	i18u			N=0~3	DBGM	
			vcchk		W:CF R:VC		VC标志检查	B	32
			vmpsw				VMP切换		
			vmpsw	LR					
			vmpintd1			W:ie	VMP切换禁止		
			vmpintd2			W:ie			
			vmpintd3			W:ie			
			vmpinte1			W:ie	VMP切换允许	A	32
			vmpinte2						
			vmpinte3						
			nop				无操作	A	16

图37

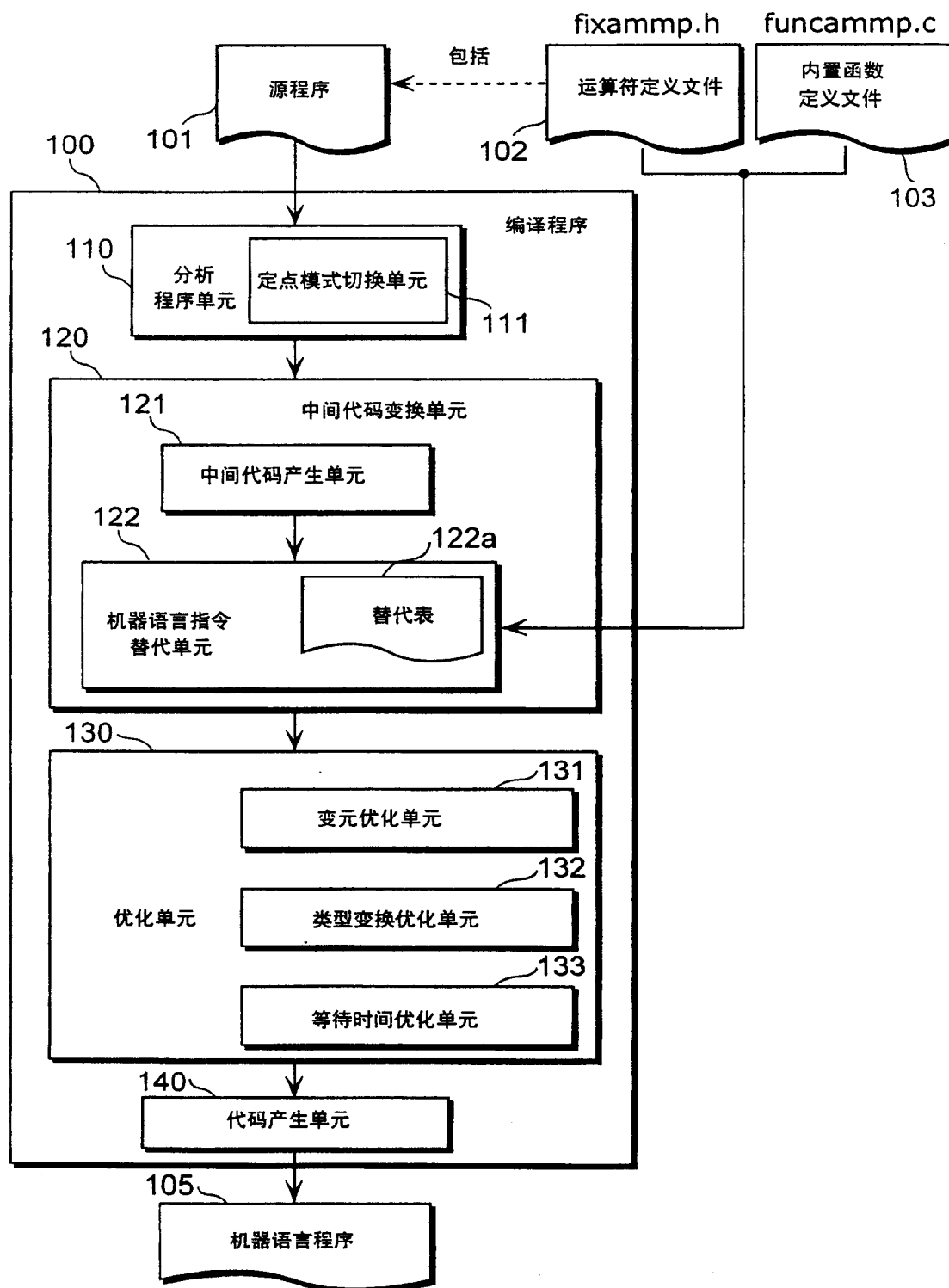


图38

```

/*****
*
* (C) Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*   fixamp.h
*   Version:
*   Release:
*   Date:      2002/6/14   v0.9.1-convertible
*
*****/

/* Avoid overloading */
#ifndef __FIXAMP__
#define __FIXAMP__

/* FIX-Lib. Class definition */
class FIX16_1;
class FIX32_1;
class FIX16_2;
class FIX32_2;

#ifdef __AMMPCC__
#pragma pack_struct
#endif // __AMMPCC__

////////////////////////////////////
//                               Member of FIX16_1
//
//                               val : Actual value in 16 bits
//
////////////////////////////////////
class FIX16_1{
    short val;
public:
    // constructor
    FIX16_1() {}
    FIX16_1(int a);
    FIX16_1(float a);
    FIX16_1(double a);
    FIX16_1(FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(volatile FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(const FIX16_1& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_1& operator=(FIX16_1 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_1& operator=(FIX16_1 a)
    {

```

图39

```

        val = a.val;
        return *this;
    }
    friend FIX16_1 operator+(FIX16_1 a);
    friend FIX16_1 operator-(FIX16_1 a);

    friend FIX16_1 operator+(FIX16_1 a, FIX16_1 b);
    friend FIX16_1 operator-(FIX16_1 a, FIX16_1 b);

    friend FIX16_1 operator*(FIX16_1 a, FIX16_1 b);
    friend FIX16_1 operator*(int a, FIX16_1 b);
    friend FIX16_1 operator*(FIX16_1 a, int b);
    friend FIX16_1 operator*(float a, FIX16_1 b);
    friend FIX16_1 operator*(FIX16_1 a, float b);
    friend FIX16_1 operator*(double a, FIX16_1 b);
    friend FIX16_1 operator*(FIX16_1 a, double b);

    friend FIX16_1 operator/(FIX16_1 a, FIX16_1 b);
    friend FIX16_1 operator/(FIX16_1 a, int b);
    friend FIX16_1 operator/(FIX16_1 a, float b);
    friend FIX16_1 operator/(FIX16_1 a, double b);

    friend FIX16_1 operator<<(FIX16_1 a, int b);
    friend FIX16_1 operator>>(FIX16_1 a, int b);

    friend bool operator<(FIX16_1 a, FIX16_1 b);
    friend bool operator>(FIX16_1 a, FIX16_1 b);
    friend bool operator<=(FIX16_1 a, FIX16_1 b);
    friend bool operator>=(FIX16_1 a, FIX16_1 b);
    friend bool operator==(FIX16_1 a, FIX16_1 b);
    friend bool operator!=(FIX16_1 a, FIX16_1 b);

    volatile FIX16_1& operator<=(int b) volatile;
        FIX16_1& operator<=(int b);
    volatile FIX16_1& operator>=(int b) volatile;
        FIX16_1& operator>=(int b);

    volatile FIX16_1& operator*=(FIX16_1 b) volatile;
        FIX16_1& operator*=(FIX16_1 b);
    volatile FIX16_1& operator*=(int b) volatile;
        FIX16_1& operator*=(int b);
    volatile FIX16_1& operator*=(float b) volatile;
        FIX16_1& operator*=(float b);
    volatile FIX16_1& operator*=(double b) volatile;
        FIX16_1& operator*=(double b);

    volatile FIX16_1& operator/=(FIX16_1 b) volatile;
        FIX16_1& operator/=(FIX16_1 b);
    volatile FIX16_1& operator/=(int b) volatile;
        FIX16_1& operator/=(int b);
    volatile FIX16_1& operator/=(float b) volatile;
        FIX16_1& operator/=(float b);
    volatile FIX16_1& operator/=(double b) volatile;
        FIX16_1& operator/=(double b);
    volatile FIX16_1& operator+=(FIX16_1 b) volatile;
        FIX16_1& operator+=(FIX16_1 b);
    volatile FIX16_1& operator-=(FIX16_1 b) volatile;
        FIX16_1& operator-=(FIX16_1 b);

```

图40

```

short value() {return val;}

// Other functions

friend FIX16_1 _fix161(short a);
friend short _bptn(FIX16_1 a);
friend FIX16_1 _fix161(FIX32_1 a);
friend float _float(FIX16_1 a);
friend double _double(FIX16_1 a);

friend FIX16_1 _abs(FIX16_1 a);
friend FIX16_1 _max(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _min(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _adds(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _subs(FIX16_1 a, FIX16_1 b);
friend int _bcnt1(FIX16_1 a);
friend int _bseq(FIX16_1 a);
friend int _bseq0(FIX16_1 a);
friend int _bseq1(FIX16_1 a);
friend FIX16_1 _round(FIX32_1 a);
friend int _extr(FIX16_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_1 a, unsigned int b, unsigned int c);

friend void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mulr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _macr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _msur(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, int b);
friend FIX16_1 _div(FIX16_1 a, float b);
friend FIX16_1 _div(FIX16_1 a, double b);

};

////////////////////////////////////
//                               Member of FIX32_1
//
//                               val : Actual value in 32 bits
//
////////////////////////////////////
class FIX32_1{
    long val;
public:
    // constructor
    FIX32_1() {}
    FIX32_1(int a);

```

图41

```

FIX32_1(float a);
FIX32_1(double a);
FIX32_1(FIX16_1 a);
FIX32_1(FIX32_1& a)
{
    val = a.val;
}
FIX32_1(volatile FIX32_1& a)
{
    val = a.val;
}
FIX32_1(const FIX32_1& a)
{
    val = a.val;
}

// Operator
volatile FIX32_1& operator=(FIX32_1 a) volatile
{
    val = a.val;
    return *this;
}
FIX32_1& operator=(FIX32_1 a)
{
    val = a.val;
    return *this;
}

friend FIX32_1 operator+(FIX32_1 a);
friend FIX32_1 operator-(FIX32_1 a);

friend FIX32_1 operator+(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator-(FIX32_1 a, FIX32_1 b);

friend FIX32_1 operator*(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator*(int a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, int b);
friend FIX32_1 operator*(float a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, float b);
friend FIX32_1 operator*(double a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, double b);

friend FIX32_1 operator/(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator/(FIX32_1 a, int b);
friend FIX32_1 operator/(FIX32_1 a, float b);
friend FIX32_1 operator/(FIX32_1 a, double b);

friend FIX32_1 operator<<(FIX32_1 a, int b);
friend FIX32_1 operator>>(FIX32_1 a, int b);

friend bool operator<(FIX32_1 a, FIX32_1 b);
friend bool operator>(FIX32_1 a, FIX32_1 b);
friend bool operator<=(FIX32_1 a, FIX32_1 b);
friend bool operator>=(FIX32_1 a, FIX32_1 b);
friend bool operator==(FIX32_1 a, FIX32_1 b);
friend bool operator!=(FIX32_1 a, FIX32_1 b);

volatile FIX32_1& operator<=<=(int b) volatile;
FIX32_1& operator<=<=(int b);
volatile FIX32_1& operator>=>=(int b) volatile;

```

图42

```

FIX32_1& operator>>=(int b):

volatile FIX32_1& operator*=(FIX32_1 b) volatile;
FIX32_1& operator*=(FIX32_1 b);
volatile FIX32_1& operator*=(int b) volatile;
FIX32_1& operator*=(int b);
volatile FIX32_1& operator*=(float b) volatile;
FIX32_1& operator*=(float b);
volatile FIX32_1& operator*=(double b) volatile;
FIX32_1& operator*=(double b);

volatile FIX32_1& operator/=(FIX32_1 b) volatile;
FIX32_1& operator/=(FIX32_1 b);
volatile FIX32_1& operator/=(int b) volatile;
FIX32_1& operator/=(int b);
volatile FIX32_1& operator/=(float b) volatile;
FIX32_1& operator/=(float b);
volatile FIX32_1& operator/=(double b) volatile;
FIX32_1& operator/=(double b);

volatile FIX32_1& operator+=(FIX32_1 b) volatile;
FIX32_1& operator+=(FIX32_1 b);
volatile FIX32_1& operator-=(FIX32_1 b) volatile;
FIX32_1& operator-=(FIX32_1 b);

long value() {return val;}

// Other functions
friend FIX32_1 _fix321(long a);
friend long _bptn(FIX32_1 a);
friend float _float(FIX32_1 a);
friend double _double(FIX32_1 a);

friend FIX32_1 _abs(FIX32_1 a);
friend FIX32_1 _max(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _min(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _adds(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _subs(FIX32_1 a, FIX32_1 b);
friend int _bcnt1(FIX32_1 a);
friend int _bseq(FIX32_1 a);
friend int _bseq0(FIX32_1 a);
friend int _bseq1(FIX32_1 a);
friend FIX16_1 _round(FIX32_1 a);

friend int _extr(FIX32_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_1 a, unsigned int b, unsigned int c);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);

friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

```

图43

```

friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, int b);
friend FIX32_1 _div(FIX32_1 a, float b);
friend FIX32_1 _div(FIX32_1 a, double b);

};

////////////////////////////////////
//                               Member of FIX16_2
//                               val : Actual value in 16 bits
//                               //////////////////////////////////
class FIX16_2{
    short val;
public:
    // constructor
    FIX16_2() {}
    FIX16_2(int a);
    FIX16_2(float a);
    FIX16_2(double a);
    FIX16_2(FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(volatile FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(const FIX16_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_2& operator=(FIX16_2 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_2& operator=(FIX16_2 a)
    {
        val = a.val;
        return *this;
    }

    friend FIX16_2 operator+(FIX16_2 a);
    friend FIX16_2 operator-(FIX16_2 a);

    friend FIX16_2 operator+(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator-(FIX16_2 a, FIX16_2 b);

    friend FIX16_2 operator*(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator*(int a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, int b);
    friend FIX16_2 operator*(float a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, float b);
    friend FIX16_2 operator*(double a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, double b);

```

图44

```

friend FIX16_2 operator/(FIX16_2 a, FIX16_2 b);
friend FIX16_2 operator/(FIX16_2 a, int b);
friend FIX16_2 operator/(FIX16_2 a, float b);
friend FIX16_2 operator/(FIX16_2 a, double b);

friend FIX16_2 operator<<(FIX16_2 a, int b);
friend FIX16_2 operator>>(FIX16_2 a, int b);

friend bool operator<(FIX16_2 a, FIX16_2 b);
friend bool operator>(FIX16_2 a, FIX16_2 b);
friend bool operator<=(FIX16_2 a, FIX16_2 b);
friend bool operator>=(FIX16_2 a, FIX16_2 b);
friend bool operator==(FIX16_2 a, FIX16_2 b);
friend bool operator!=(FIX16_2 a, FIX16_2 b);

volatile FIX16_2& operator<=(int b) volatile;
FIX16_2& operator<=(int b);
volatile FIX16_2& operator>=(int b) volatile;
FIX16_2& operator>=(int b);

volatile FIX16_2& operator*=(FIX16_2 b) volatile;
FIX16_2& operator*=(FIX16_2 b);
volatile FIX16_2& operator*=(int b) volatile;
FIX16_2& operator*=(int b);
volatile FIX16_2& operator*=(float b) volatile;
FIX16_2& operator*=(float b);
volatile FIX16_2& operator*=(double b) volatile;
FIX16_2& operator*=(double b);

volatile FIX16_2& operator/=(FIX16_2 b) volatile;
FIX16_2& operator/=(FIX16_2 b);
volatile FIX16_2& operator/=(int b) volatile;
FIX16_2& operator/=(int b);
volatile FIX16_2& operator/=(float b) volatile;
FIX16_2& operator/=(float b);
volatile FIX16_2& operator/=(double b) volatile;
FIX16_2& operator/=(double b);

volatile FIX16_2& operator+=(FIX16_2 b) volatile;
FIX16_2& operator+=(FIX16_2 b);
volatile FIX16_2& operator-=(FIX16_2 b) volatile;
FIX16_2& operator-=(FIX16_2 b);

short value0 {return val;}

// Other functions
friend FIX16_2 _fix162(short a);
friend short _bptn(FIX16_2 a);
friend FIX16_2 _fix162(FIX32_2 a);
friend float _float(FIX16_2 a);
friend double _double(FIX16_2 a);

friend FIX16_2 _abs(FIX16_2 a);
friend FIX16_2 _max(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _min(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _adds(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _subs(FIX16_2 a, FIX16_2 b);

```

图45

```

friend int      _bcnt1(FIX16_2 a);
friend int      _bseq(FIX16_2 a);
friend int      _bseq0(FIX16_2 a);
friend int      _bseq1(FIX16_2 a);
friend FIX16_2  _round(FIX32_2 a);

friend int _extr(FIX16_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_2 a, unsigned int b, unsigned int c);

friend void      _mul(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void      _msu(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend FIX16_2   _div(FIX16_2 a, FIX16_2 b);
friend FIX16_2   _div(FIX16_2 a, int b);
friend FIX16_2   _div(FIX16_2 a, float b);
friend FIX16_2   _div(FIX16_2 a, double b);

};

////////////////////////////////////
//                               Member of FIX32_2
//
//                               val : Actual value in 32 bits
//
////////////////////////////////////
class FIX32_2{
    long val;
public:
    // constructor
    FIX32_2() {}
    FIX32_2(int a);
    FIX32_2(float a);
    FIX32_2(double a);
    FIX32_2(FIX16_2 a);
    FIX32_2(FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(volatile FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(const FIX32_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX32_2& operator=(FIX32_2 a) volatile
    {

```


图46

```

        val = a.val;
        return *this;
    }
    FIX32_2& operator=(FIX32_2 a)
    {
        val = a.val;
        return *this;
    }
    friend FIX32_2 operator+(FIX32_2 a);
    friend FIX32_2 operator-(FIX32_2 a);

    friend FIX32_2 operator+(FIX32_2 a, FIX32_2 b);
    friend FIX32_2 operator-(FIX32_2 a, FIX32_2 b);

    friend FIX32_2 operator*(FIX32_2 a, FIX32_2 b);
    friend FIX32_2 operator*(int a, FIX32_2 b);
    friend FIX32_2 operator*(FIX32_2 a, int b);
    friend FIX32_2 operator*(float a, FIX32_2 b);
    friend FIX32_2 operator*(FIX32_2 a, float b);
    friend FIX32_2 operator*(double a, FIX32_2 b);
    friend FIX32_2 operator*(FIX32_2 a, double b);

    friend FIX32_2 operator/(FIX32_2 a, FIX32_2 b);
    friend FIX32_2 operator/(FIX32_2 a, int b);
    friend FIX32_2 operator/(FIX32_2 a, float b);
    friend FIX32_2 operator/(FIX32_2 a, double b);

    friend FIX32_2 operator<<(FIX32_2 a, int b);
    friend FIX32_2 operator>>(FIX32_2 a, int b);

    friend bool operator<(FIX32_2 a, FIX32_2 b);
    friend bool operator>(FIX32_2 a, FIX32_2 b);
    friend bool operator<=(FIX32_2 a, FIX32_2 b);
    friend bool operator>=(FIX32_2 a, FIX32_2 b);
    friend bool operator==(FIX32_2 a, FIX32_2 b);
    friend bool operator!=(FIX32_2 a, FIX32_2 b);

    volatile FIX32_2& operator<=(int b) volatile;
        FIX32_2& operator<=(int b);
    volatile FIX32_2& operator>=(int b) volatile;
        FIX32_2& operator>=(int b);

    volatile FIX32_2& operator*=(FIX32_2 b) volatile;
        FIX32_2& operator*=(FIX32_2 b);
    volatile FIX32_2& operator*=(int b) volatile;
        FIX32_2& operator*=(int b);
    volatile FIX32_2& operator*=(float b) volatile;
        FIX32_2& operator*=(float b);
    volatile FIX32_2& operator*=(double b) volatile;
        FIX32_2& operator*=(double b);

    volatile FIX32_2& operator/=(FIX32_2 b) volatile;
        FIX32_2& operator/=(FIX32_2 b);
    volatile FIX32_2& operator/=(int b) volatile;
        FIX32_2& operator/=(int b);
    volatile FIX32_2& operator/=(float b) volatile;
        FIX32_2& operator/=(float b);
    volatile FIX32_2& operator/=(double b) volatile;
        FIX32_2& operator/=(double b);

```

图47

```

volatile FIX32_2& operator+=(FIX32_2 b) volatile;
    FIX32_2& operator+=(FIX32_2 b);
volatile FIX32_2& operator-=(FIX32_2 b) volatile;
    FIX32_2& operator-=(FIX32_2 b);

long value() {return val;}

// Other functions
friend FIX32_2 _fix322(long a);
friend long _bptn(FIX32_2 a);
friend float _float(FIX32_2 a);
friend double _double(FIX32_2 a);

friend FIX32_2 _abs(FIX32_2 a);
friend FIX32_2 _max(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _min(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _adds(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _subs(FIX32_2 a, FIX32_2 b);
friend int _bcnt1(FIX32_2 a);
friend int _bseq(FIX32_2 a);
friend int _bseq0(FIX32_2 a);
friend int _bseq1(FIX32_2 a);
friend FIX16_2 _round(FIX32_2 a);

friend int _extr(FIX32_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_2 a, unsigned int b, unsigned int c);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);

friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, int b);
friend FIX32_2 _div(FIX32_2 a, float b);
friend FIX32_2 _div(FIX32_2 a, double b);

};

#if defined(__AMMPCC__)
#pragma pack_struct_default
#endif //__AMMPCC__

// other functions
#if defined(__AMMPCC__)
#pragma _enable_asm_begin

static inline FIX16_1 _fix161(FIX32_1 a)
{
    FIX16_1 result;

```

图48

```
asm(vr0 = a) {
    asr vr1, vr0, 16;
} (result = vr1);

return result;
}

static inline FIX16_2 _fix162(FIX32_2 a)
{
    FIX16_2 result;

    asm(vr0 = a) {
        asr vr1, vr0, 16;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _fix161(short a)
{
    FIX16_1 result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_2 _fix162(short a)
{
    FIX16_2 result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX32_1 _fix321(long a)
{
    FIX32_1 result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX32_2 _fix322(long a)
{
    FIX32_2 result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);
```

图49

```
    return result;
}

static inline short _bptn(FIX16_1 a)
{
    short result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline short _bptn(FIX16_2 a)
{
    short result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline long _bptn(FIX32_1 a)
{
    long result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline long _bptn(FIX32_2 a)
{
    long result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _abs(FIX16_1 a)
{
    FIX16_1 result;

    asm(vr0 = a) {
        absvh vr1, vr0;
    } (result = vr1);

    return result;
}
```

图50

```
static inline FIX32_1 _abs(FIX32_1 a)
{
    FIX32_1 result;

    asm(vr0 = a) {
        absvw vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_2 _abs(FIX16_2 a)
{
    FIX16_2 result;

    asm(vr0 = a) {
        absvh vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX32_2 _abs(FIX32_2 a)
{
    FIX32_2 result;

    asm(vr0 = a) {
        absvw vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _max(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b) {
        max vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX32_1 _max(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        max vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX16_2 _max(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;
```

图51

```
asm(vr0 = a, vr1 = b) {  
    max vr2, vr0, vr1;  
}(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _max(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        max vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_1 _min(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_1 _min(FIX32_1 a, FIX32_1 b)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_2 _min(FIX16_2 a, FIX16_2 b)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _min(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;
```

图52

```
    } (result = vr2);  
    return result;  
}  
  
static inline FIX16_1 _adds(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX32_1 _adds(FIX32_1 a, FIX32_1 b)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX16_2 _adds(FIX16_2 a, FIX16_2 b)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _adds(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX16_1 _subs(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        subs    vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}
```

图53

```
static inline FIX32_1 _subs(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        subs    vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX16_2 _subs(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        subs    vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX32_2 _subs(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        subs    vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline int _bcnt1(FIX16_1 a)
{
    int result;

    asm(vr0 = a) {
        bcnt1    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bcnt1(FIX16_2 a)
{
    int result;

    asm(vr0 = a) {
        bcnt1    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bcnt1(FIX32_1 a)
{
    int result;
```


图54

```
asm(vr0 = a) {  
    bcnt1    vr1, vr0;  
} (result = vr1);  
  
    return result;  
}  
  
static inline int _bcnt1(FIX32_2 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bcnt1    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq     vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq0(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq0    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq1(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq1    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq(FIX32_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq     vr1, vr0;  
    } (result = vr1);
```

图55

```
    return result;
}

static inline int _bseq0(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq0    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq1    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq     vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq0(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq0    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq1    vr1, vr0;
    }(result = vr1);

    return result;
}
```

图56

```
static inline int _bseq(FIX32_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq0(FIX32_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq0   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq1(FIX32_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq1   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _round(FIX32_1 a)
{
    FIX16_1 result;

    asm(vr0 = a) {
        rndvh   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_2 _round(FIX32_2 a)
{
    FIX16_2 result;

    asm(vr0 = a) {
        rndvh   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0, vr2, vr0, vr1;
    }
```

图57

```

    } (c = vr2);
}

static inline void _mulr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0, vr2, vr1, vr0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

图58

```

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr1,vr0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr0,vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr0,vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {

```

图59

```

    fmachww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmacww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmacww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _macr(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachhr m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

图60

```

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmsuw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmsuw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msur(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhr m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

inline FIX16_1 operator/(FIX16_1 a, FIX16_1 b)
{

```

图61

```

FIX16_1 result:

asm(vr0 = a, vr1 = b) {
    extw m0, vr3, vr0;
    aslp m0, vr4, mh0, vr3, 15;
    div  mh1, vr5, mh0, vr4, vr1;
    sath vr2, vr5;
} (result = vr2);

return result;
}

inline FIX16_2 operator/(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result:

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        aslp m0, vr4, mh0, vr3, 14;
        div  mh1, vr5, mh0, vr4, vr1;
        sath vr2, vr5;
    } (result = vr2);

    return result;
}

inline FIX32_1 operator/(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result:

    asm(vr0 = a, vr1 = b) {
        mskgen    vr3, 31, 31; //vr3 = 0x80000000
        cmpeq     C0:C1, vr1, vr3;
        [C0]      negvw    vr2, vr0; //in the case of vr1==(-1), sign-change
        cmpgtn    C2:C3, vr0, 0, C1; //in the case of vr1!=(-1)
        [C2]      negvw    vr4, vr0; //sign-change vr0 negatively (vr0')
        [C3]      mov      vr4, vr0;
        [C2]      cmpgtn    C2:C3, vr1, 0, C1;
        [C2]      negvw    vr5, vr1; //sign-change vr1 negatively (vr1')
        [C3]      mov      vr5, vr1;
        cmplen    C2:C3, vr4, vr5, C1;
        [C2]      lsr      vr6, vr0, 31; //vr0' <= 2vr1' in the case of (ovf decision),
over flow
        [C2]      lsr      vr7, vr1, 31;
        [C2]      xor      vr8, vr6, vr7;
        cmpeqn    C4:C5, vr8, 0, C2;
        [C4]      mskgen    vr2, 30, 0; //vr2 = 0x7fffffff
        [C5]      mskgen    vr2, 31, 31; //vr2 = 0x80000000
        [C3]      extw      m0, vr9, vr0; //in the case of dividend<divider
        [C3]      aslp      m0, vr10, mh0, vr9, 31;
        [C3]      div       mh1, vr2, mh0, vr10, vr1;
    } (result = vr2);

    return result;
}

```


图62

```

inline FIX32_2 operator/(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen    vr3, 31, 31;    //vr3 = 0x80000000
        mskgen    vr4, 31, 30;    //vr4 = 0xc0000000
        cmpgt     C0:C1, vr0, 0;
        [C0]      negvw    vr5, vr0;    //sign-change vr0 negatively (vr0')
        [C1]      mov      vr5, vr0;
        cmpgt     C0:C1, vr1, 0;
        [C0]      negvw    vr6, vr1;    //sign-change vr1 negatively (vr1')
        [C1]      mov      vr6, vr1;
        cmpeq     C0:C1, vr0, vr3;
        cmplta    C0:C1, vr6, vr4, C0; //vr0=-2 and vr1' < 0xc0000000?
        [C1]      aslvw    vr6, vr6, 1;    //!when (vr0=-2 and vr1' < 0xc0000000)
        cmplea    C2:C3, vr5, vr6, C1;
        [C2]      lsr      vr7, vr0, 31; //!in the case of (vr0=-2 and vr1'
        [C2]      lsr      vr8, vr1, 31; //< 0xc0000000) and vr0' <= 2vr1'
        [C2]      xor      vr9, vr7, vr8;    //(ovf decision), overflow
        cmpeqn    C4:C5, vr9, 0, C2;
        [C4]      mskgen    vr2, 30, 0;    //vr2 = 0x7fffffff
        [C5]      mskgen    vr2, 31, 31;    //vr2 = 0x80000000
        [C3]      extw     m0, vr10, vr0;    //other than that
        [C3]      aslp     m0, vr11, mh0, vr10, 30;
        [C3]      div      mh1, vr2, mh0, vr11, vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, int b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        div  mh1, vr4, mh0, vr3, vr1;
        sath vr2, vr4;
    } (result = vr2);

    return result;
}

inline FIX16_2 operator/(FIX16_2 a, int b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        div  mh1, vr4, mh0, vr3, vr1;
        sath vr2, vr4;
    } (result = vr2);

    return result;
}

```

图63

```

inline FIX32_1 operator/(FIX32_1 a, int b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        cmpeq    C0:C1, vr1, 1;
        [C0]    mov     vr2, vr0;    //when dividend=1, Rc = Ra
        [C1]    mskgen   vr3, 31, 31; //0x80000000
        cmpeqn   C2:C3, vr0, vr3, C1;
        cmpeqa   C4:C5, vr1, -1, C2;
        [C4]    mskgen   vr2, 30, 0; //when dividend=-1 and divisor=-1, Rc =
0x7fffffff
        [C5]    extw     m0, vr4, vr0; //other than that
        [C5]    div      mh1, vr2, mh0, vr4, vr1;
    } (result = vr2);

    return result;
}

inline FIX32_2 operator/(FIX32_2 a, int b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen   vr3, 31, 31; //Rd = 0x80000000
        mskgen   vr4, 31, 30; //Re = 0xc0000000
        cmpgt    C0:C1, vr0, 0;
        [C0]    negvw    vr5, vr0; //sign-change Ra negatively (Ra')
        [C1]    mov      vr5, vr0;
        aslvw    vr6, vr1, 30; //int-->FIX32_2
        cmpgt    C0:C1, vr6, 0;
        [C0]    negvw    vr7, vr6; //sign-change Rb negatively (Rb')
        [C1]    mov      vr7, vr6;
        cmpeq    C0:C1, vr0, vr3;
        cmplt    C0:C1, vr7, vr4, C0; //Ra=-2 and Rb' < 0xc0000000 ?
        [C1]    aslvw    vr7, vr7, 1; //!when (Ra=-2 and Rb' < 0xc0000000)
        cmplea   C2:C3, vr5, vr7, C1;
        [C2]    lsr      vr8, vr0, 31; //!in the case of (Ra=-2 and Rb' < 0xc0000000)
        [C2]    lsr      vr9, vr1, 31; // and Ra' <= 2Rb' (ovf decision)
        [C2]    xor      vr10, vr8, vr9; //overflow
        cmpeqn   C4:C5, vr10, 0, C2;
        [C4]    mskgen   vr2, 30, 0; //Rc = 0x7fffffff
        [C5]    mskgen   vr2, 31, 31; //Rc = 0x80000000
        [C3]    extw     m0, vr11, vr0; //other than that
        [C3]    div      mh1, vr2, mh0, vr11, vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, float b)
{
    FIX16_1 c = b;
    return a/c;
}

```

图64

```
inline FIX16_1 operator/(FIX16_1 a, double b)
{
    FIX16_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, float b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, double b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, float b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, double b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, float b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, double b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX16_1& FIX16_1::operator/=(FIX16_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(FIX16_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}
```

图65

```
inline volatile FIX16_1& FIX16_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(FIX32_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(FIX32_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}
```

图66

```

}

inline volatile FIX32_1& FIX32_1::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(FIX16_2 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(FIX16_2 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(double b)
{
    *this = *this / b;

```

图67

```

    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(FIX32_2 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(FIX32_2 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

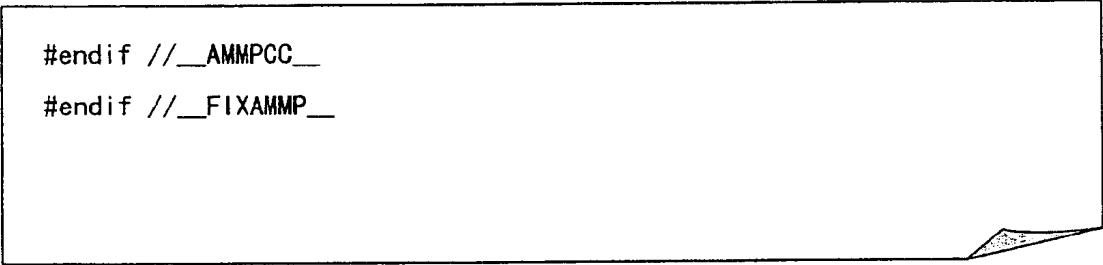
inline FIX32_2& FIX32_2::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

#pragma _enable_asm_end

```

图68



```
#endif //__AMMPCC__  
#endif //__FIXAMMP__
```

图69

```

/*****
*
* (C) Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*   funcammp.h
*   Version:
*   Release:
*   Date:      2002/6/19
*
*****/

/* Avoid overloading */
#ifndef __FUNCAMMP__
#define __FUNCAMMP__

#include <stdlib.h>

#ifndef __AMMPCC__

long _abs(long);
long _max(long, long);
long _min(long, long);
long _adds(long, long);
long _subs(long, long);
int _bcnt1(long);
int _bseq0(long);
int _bseq1(long);
int _bseq(long);
int _log2(size_t size);
long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
void _clrm(long&, long&);
void _mul(long&, long&, long&, long, long);
void _mac(long&, long&, long&, long, long);
void _msu(long&, long&, long&, long, long);
void *_modulo_add(void *, int, int, size_t, void *);
void *_brev_add(void *, int, int, int, size_t, void *);

#else /* defined __AMMPCC__ */

#pragma _enable_asm_begin
#pragma _enable_inline_begin

long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
int _log2(size_t size);

static inline long
_abs(long data)
{
    long result;

    asm(vr0 = data) {
        abs    vr1, vr0;
    } (result = vr1);

    return result;
}

```


图70

```
static inline long
_max(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        max vr2, vr1, vr0;
    } (result = vr2);

    return result;
}

static inline long
_min(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        min vr2, vr1, vr0;
    } (result = vr2);

    return result;
}

static inline long
_adds(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        adds vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline long
_subs(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        subs vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline int
_bcntl(long data)
{
    long result;

    asm(vr0 = data) {
        bcntl vr1, vr0;
    } (result = vr1);

    return result;
}
```

图71

```

static inline int
_bseq0(long data)
{
    long result;

    asm(vr0 = data) {
        bseq0 vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int
_bseq1(long data)
{
    long result;

    asm(vr0 = data) {
        bseq1 vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int
_bseq(long data)
{
    long result;

    asm(vr0 = data) {
        bseq vr1, vr0;
    } (result = vr1);

    return result;
}

static inline void
_clrm(long &mh, long &ml)
{
    asm() {
        mul m0, vr1, vr0, 0;
    } (mh = mh0, ml = ml0);
}

static inline void
_mul(long &mh, long &ml, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b) {
        mul m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void
_mac(long &mh, long &ml, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        mac m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

图72

```

}

static inline void
_msu(long &mh, long &ml, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        msu    m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void *
_modulo_add(void *addr, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = imm << tmp1;

    asm(vr0 = addr, vr2 = base, vr3 = tmp2, vr4 = tmp3) {
        mov     CFR0, vr3;
        add     vr6, vr0, vr4;
        addmsk  vr7, vr2, vr6;
    } (p = vr7);

    return p;
}

static inline void *
_brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3, tmp4;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = 16 - mask - tmp1;
    tmp4 = imm << tmp3;

    asm(vr0 = addr, vr1 = cnt, vr2 = base, vr3 = tmp2, vr4 = tmp3, vr5 = tmp4) {
        mov     CFR0, vr3;
        lsl     vr6, vr1, vr4;
        add     vr7, vr6, vr5;
        mskbrvh vr8, vr2, vr7;
    } (p = vr8);

    return p;
}

#pragma _enable_inline_end
#pragma _enable_asm_end

#endif /* __AMMPCC__ */

#endif /* __FUNCAMMP__ */

```

图73

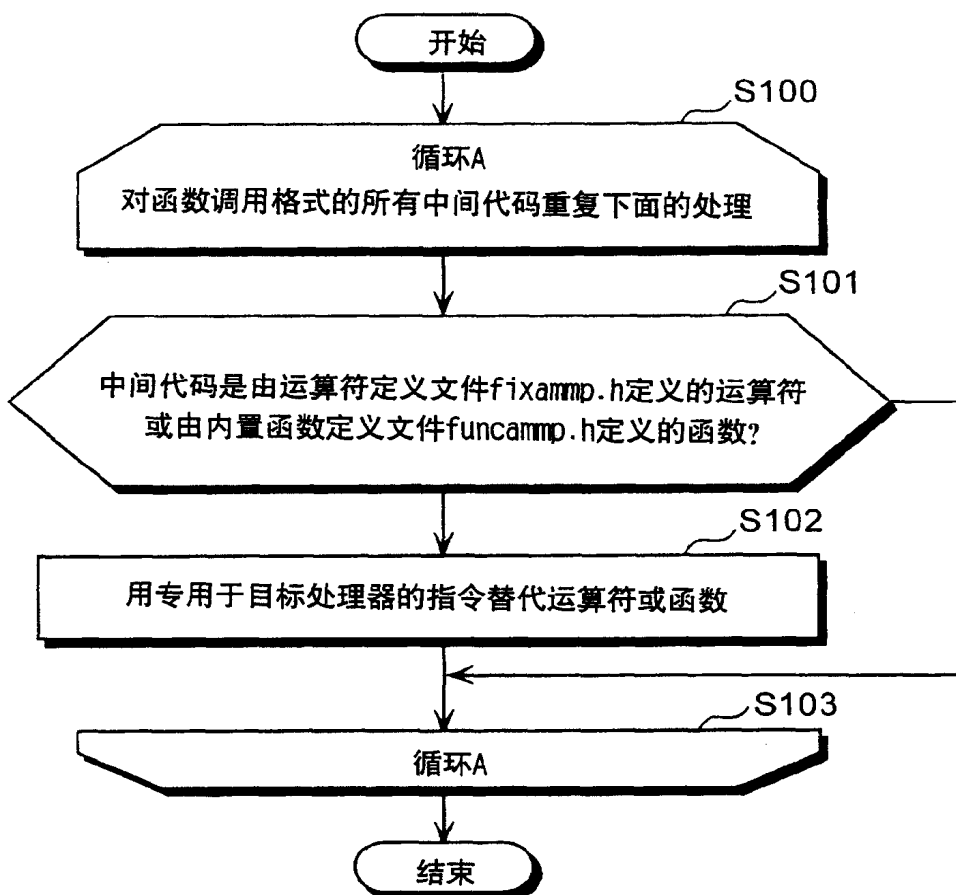


图74

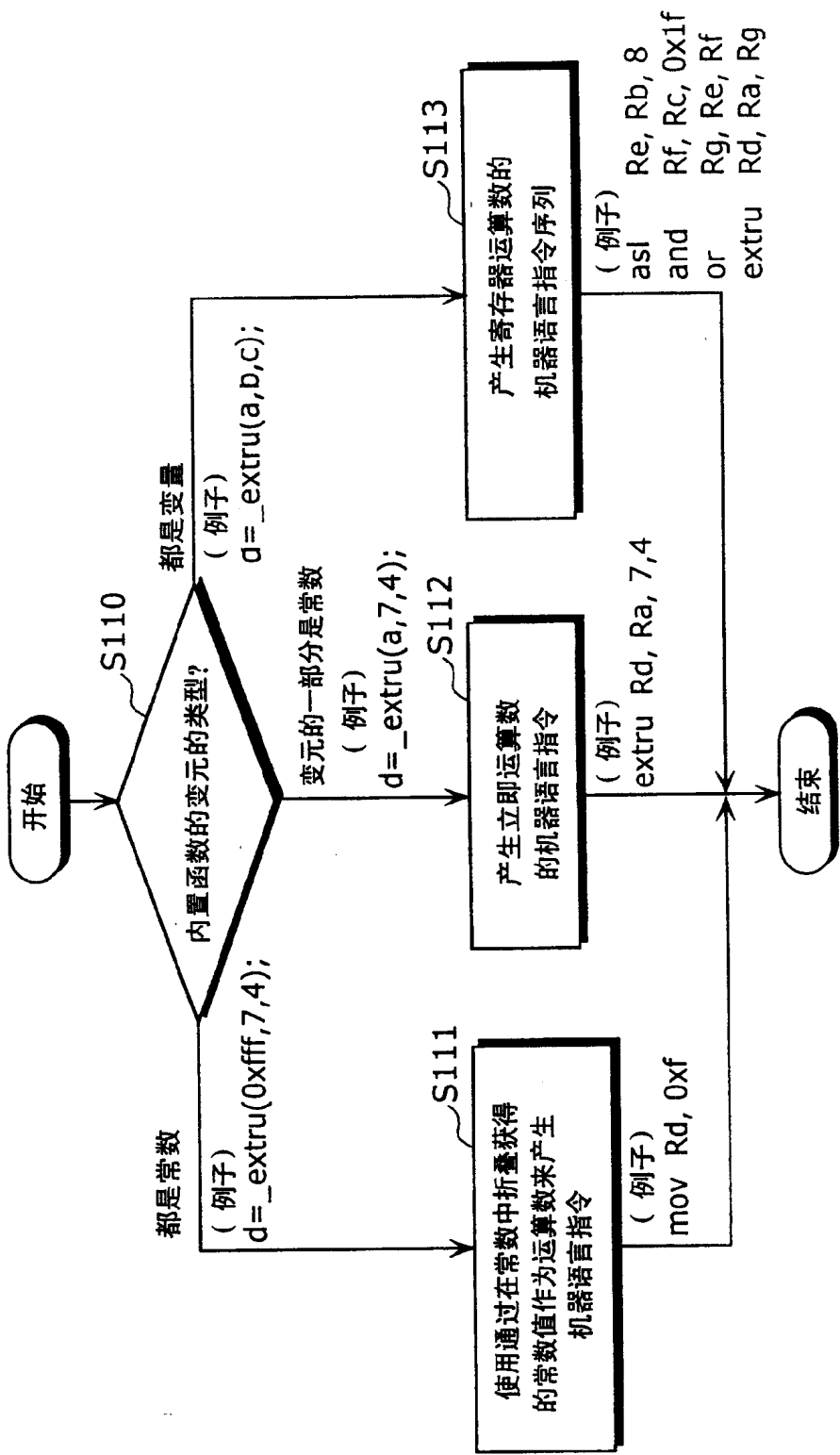


图75A

普通算术树

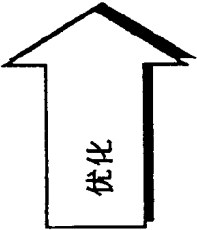
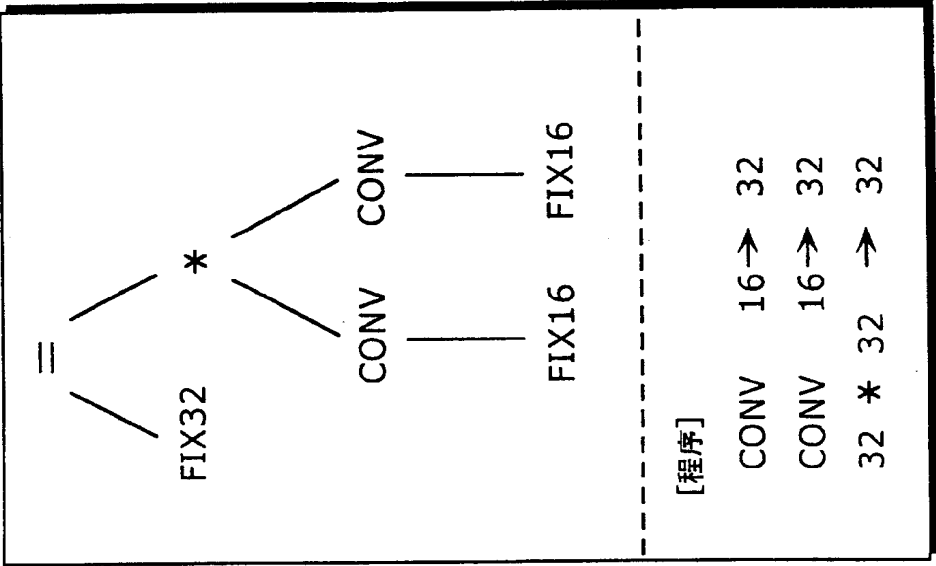


图75B

优化之后的算术树

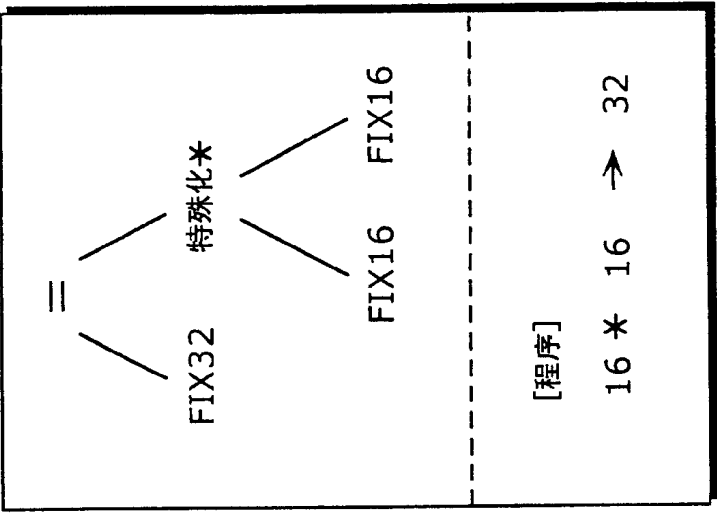


图76A

配置等待时间的例子1

<Example> Leave a space of 2 cycles between wte and rde

```
static inline int getbits(int a)
{
    int result ;
    asm (vr0 = a) {
        LATENCY L1, L2, 2 ;
        mov vr1, AVL_BASEADDR ;
L1:      wte C0:C1, (vr1, AVL_GETBITS), vr0 ;
L2:      rde C0:C1, vr2, (vr1, AVL_READPORT) ;
    } (result = vr2) ;
    return result ;
}
```

图76B

配置等待时间的例子2

<Example> Leave a space of 2 cycles after wte

until next extended register access

```
static inline void skipbits(int a)
{
    asm (vr0 = a) {
        mov vr1, AVL_BASEADDR ;
        wte C0:C1, (vr1, AVL_SKIPBITS), vr0, LATENCY(2) ;
    } ;
}
```

图77A

采样程序

```

#pragma _save_fxpmode func
func(void)
{
    FIX16_1 a;
    (Main body)
}

```

Save FIX-type mode
Configure FIX-type mode to _lsystem
→ (Main body)
Return to FIX-type mode

图77B

内部处理

```

Save:  mov vr0,PSR0
1Configuration: or   vr1,vr0,0x20 + mov PSR0,vr1
0Configuration: andn vr1,vr0,0x20 + mov PSR0,vr1
Return: mov PSR0,vr0

```

图77C

应用例子

关于四个函数f11、f21、f22和f23
在函数f11:_1系统调用函数f21:_2系统;
函数f21:_2系统调用函数f22:_2系统;
函数f22:_2系统调用函数f23:_2系统的情况下,

由于可以由其他模式调用的唯一的函数是f21, 所以可以通过执行只对这个函数执行编译指示指定来切换到一个正常模式。

图78A

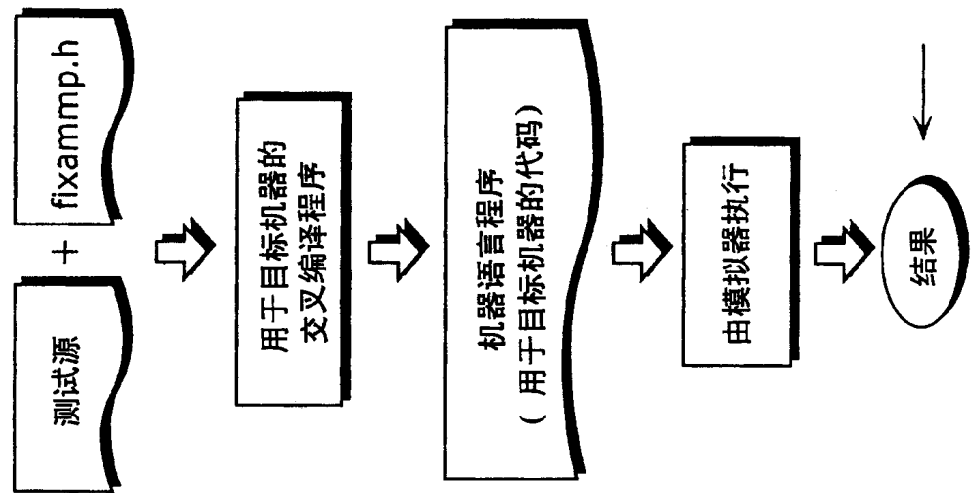


图78B

